



# PYTHON API Reference



# Table of contents

<b>1. Introduction</b>	<b>1</b>
<b>2. Using the Yocto-Demo with Python</b>	<b>3</b>
2.1. Source files	3
2.2. Dynamic library	3
2.3. Control of the Led function	3
2.4. Control of the module part	5
2.5. Error handling	7
Blueprint	10
<b>3. Reference</b>	<b>10</b>
3.1. General functions	11
3.2. Accelerometer function interface	36
3.3. AnButton function interface	82
3.4. CarbonDioxide function interface	124
3.5. ColorLed function interface	167
3.6. Compass function interface	200
3.7. Current function interface	244
3.8. DataLogger function interface	287
3.9. Formatted data sequence	322
3.10. Recorded data sequence	332
3.11. Unformatted data sequence	345
3.12. Digital IO function interface	360
3.13. Display function interface	408
3.14. DisplayLayer object interface	459
3.15. External power supply control interface	491
3.16. Files function interface	520
3.17. GenericSensor function interface	553
3.18. Gyroscope function interface	603
3.19. Yocto-hub port interface	658
3.20. Humidity function interface	687
3.21. Led function interface	730
3.22. LightSensor function interface	761
3.23. Magnetometer function interface	805
3.24. Measured value	851

3.25. Module control interface .....	857
3.26. Network function interface .....	903
3.27. OS control .....	964
3.28. Power function interface .....	991
3.29. Pressure function interface .....	1038
3.30. Pwm function interface .....	1081
3.31. PwmPowerSource function interface .....	1123
3.32. Quaternion interface .....	1150
3.33. Real Time Clock function interface .....	1193
3.34. Reference frame configuration .....	1224
3.35. Relay function interface .....	1264
3.36. Sensor function interface .....	1304
3.37. Servo function interface .....	1347
3.38. Temperature function interface .....	1386
3.39. Tilt function interface .....	1431
3.40. Voc function interface .....	1474
3.41. Voltage function interface .....	1517
3.42. Voltage source function interface .....	1560
3.43. WakeUpMonitor function interface .....	1596
3.44. WakeUpSchedule function interface .....	1635
3.45. Watchdog function interface .....	1676
3.46. Wireless function interface .....	1725

<b>Index .....</b>	<b>1759</b>
--------------------	-------------

# 1. Introduction

This manual is intended to be used as a reference for Yoctopuce Python library, in order to interface your code with USB sensors and controllers.

The next chapter is taken from the free USB device Yocto-Demo, in order to provide a concrete examples of how the library is used within a program.

The remaining part of the manual is a function-by-function, class-by-class documentation of the API. The first section describes all general-purpose global function, while the forthcoming sections describe the various classes that you may have to use depending on the Yoctopuce device being used. For more informations regarding the purpose and the usage of a given device attribute, please refer to the extended discussion provided in the device-specific user manual.



## 2. Using the Yocto-Demo with Python

Python is an interpreted object oriented language developed by Guido van Rossum. Among its advantages is the fact that it is free, and the fact that it is available for most platforms, Windows as well as UNIX. It is an ideal language to write small scripts on a napkin. The Yoctopuce library is compatible with Python 2.6+ and 3+. It works under Windows, Mac OS X, and Linux, Intel as well as ARM. The library was tested with Python 2.6 and Python 3.2. Python interpreters are available on the Python web site<sup>1</sup>.

### 2.1. Source files

The Yoctopuce library classes<sup>2</sup> for Python that you will use are provided as source files. Copy all the content of the *Sources* directory in the directory of your choice and add this directory to the *PYTHONPATH* environment variable. If you use an IDE to program in Python, refer to its documentation to configure it so that it automatically finds the API source files.

### 2.2. Dynamic library

A section of the low-level library is written in C, but you should not need to interact directly with it: it is provided as a DLL under Windows, as a *.so* files under UNIX, and as a *.dylib* file under Mac OS X. Everything was done to ensure the simplest possible interaction from Python: the distinct versions of the dynamic library corresponding to the distinct operating systems and architectures are stored in the *cdll* directory. The API automatically loads the correct file during its initialization. You should not have to worry about it.

If you ever need to recompile the dynamic library, its complete source code is located in the Yoctopuce C++ library.

In order to keep them simple, all the examples provided in this documentation are console applications. Naturally, the libraries function in a strictly identical manner if you integrate them in an application with a graphical interface.

### 2.3. Control of the Led function

A few lines of code are enough to use a Yocto-Demo. Here is the skeleton of a Python code snippet to use the Led function.

---

<sup>1</sup> <http://www.python.org/download/>

<sup>2</sup> [www.yoctopuce.com/EN/libraries.php](http://www.yoctopuce.com/EN/libraries.php)

```
[...]

errmsg=YRefParam()
#Get access to your device, connected locally on USB for instance
YAPI.RegisterHub("usb",errmsg)
led = YLed.FindLed("YCTOPOC1-123456.led")

# Hot-plug is easy: just check that the device is online
if led.isOnline():
    #Use led.set_power()
    ...

[...]
```

Let's look at these lines in more details.

## YAPI.RegisterHub

The `yAPI.RegisterHub` function initializes the Yoctopuce API and indicates where the modules should be looked for. When used with the parameter "usb", it will use the modules locally connected to the computer running the library. If the initialization does not succeed, this function returns a value different from `YAPI.SUCCESS` and `errmsg` contains the error message.

## YLed.FindLed

The `YLed.FindLed` function allows you to find a led from the serial number of the module on which it resides and from its function name. You can use logical names as well, as long as you have initialized them. Let us imagine a Yocto-Demo module with serial number `YCTOPOC1-123456` which you have named "MyModule", and for which you have given the `led` function the name "MyFunction". The following five calls are strictly equivalent, as long as "MyFunction" is defined only once.

```
led = YLed.FindLed("YCTOPOC1-123456.led")
led = YLed.FindLed("YCTOPOC1-123456.MyFunction")
led = YLed.FindLed("MyModule.led")
led = YLed.FindLed("MyModule.MyFunction")
led = YLed.FindLed("MyFunction")
```

`YLed.FindLed` returns an object which you can then use at will to control the led.

## isOnline

The `isOnline()` method of the object returned by `YLed.FindLed` allows you to know if the corresponding module is present and in working order.

## set\_power

The `set_power()` function of the object returned by `YLed.FindLed` allows you to turn on and off the led. The argument is `YLed.POWER_ON` or `YLed.POWER_OFF`. In the reference on the programming interface, you will find more methods to precisely control the luminosity and make the led blink automatically.

## A real example

Launch Python and open the corresponding sample script provided in the directory **Examples/Doc-GettingStarted-Yocto-Demo** of the Yoctopuce library.

In this example, you will recognize the functions explained above, but this time used with all side materials needed to make it work nicely as a small demo.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os,sys
from yocto_api import *
from yocto_led import *

def usage():
    scriptname = os.path.basename(sys.argv[0])
    print("Usage:")
```



```

print(scriptname+' <serial_number>')
print(scriptname+' <logical_name>')
print(scriptname+' any ')
sys.exit()

def die(msg):
    sys.exit(msg+' (check USB cable)')

def setLedState(led,state):
    if led.isOnline():
        if state :
            led.set_power(YLed.POWER_ON)
        else:
            led.set_power(YLed.POWER_OFF)
    else:
        print('Module not connected (check identification and USB cable)')

errmsg=YRefParam()

if len(sys.argv)<2 : usage()

target=sys.argv[1]

# Setup the API to use local USB devices
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("init error"+errmsg.value)

if target=='any':
    # retrieve any RGB led
    led = YLed.FirstLed()
    if led is None :
        die('No module connected')
else:
    led= YLed.FindLed(target + '.led')

if not(led.isOnline()):die('device not connected')

print('0: turn test led OFF')
print('1: turn test led ON')
print('x: exit')

try: input = raw_input # python 2.x fix
except: pass

c= input("command:")

while c!='x':
    if c=='0' : setLedState(led,False);
    elif c=='1' :setLedState(led,True);
    c= input("command:")

```

## 2.4. Control of the module part

Each module can be controlled in a similar manner, you can find below a simple sample program displaying the main parameters of the module and enabling you to activate the localization beacon.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
import os,sys
from yocto_api import *

def usage():
    sys.exit("usage: demo <serial or logical name> [ON/OFF]")

errmsg =YRefParam()
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("RegisterHub error: " + str(errmsg))

if len(sys.argv)<2 : usage()

m = YModule.FindModule(sys.argv[1]) ## use serial or logical name

```

```

if m.isOnline():
    if len(sys.argv) > 2:
        if sys.argv[2].upper() == "ON" : m.set_beacon(YModule.BEACON_ON)
        if sys.argv[2].upper() == "OFF" : m.set_beacon(YModule.BEACON_OFF)

    print("serial:      " + m.get_serialNumber())
    print("logical name: " + m.get_logicalName())
    print("luminosity:    " + str(m.get_luminosity()))
    if m.get_beacon() == YModule.BEACON_ON:
        print("beacon:      ON")
    else:
        print("beacon:      OFF")
    print("upTime:      " + str(m.get_upTime()/1000)+" sec")
    print("USB current:  " + str(m.get_usbCurrent())+" mA")
    print("logs:\n" + m.get_lastLogs())
else:
    print(sys.argv[1] + " not connected (check identification and USB cable)")

```

Each property `xxx` of the module can be read thanks to a method of type `YModule.get_xxxx()`, and properties which are not read-only can be modified with the help of the `YModule.set_xxx()` method. For more details regarding the used functions, refer to the API chapters.

## Changing the module settings

When you want to modify the settings of a module, you only need to call the corresponding `YModule.set_xxx()` function. However, this modification is performed only in the random access memory (RAM) of the module: if the module is restarted, the modifications are lost. To memorize them persistently, it is necessary to ask the module to save its current configuration in its permanent memory. To do so, use the `YModule.saveToFlash()` method. Inversely, it is possible to force the module to forget its current settings by using the `YModule.revertFromFlash()` method. The short example below allows you to modify the logical name of a module.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
import os,sys
from yocto_api import *

def usage():
    sys.exit("usage: demo <serial or logical name> <new logical name>")

if len(sys.argv) != 3 : usage()

errmsg =YRefParam()
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("RegisterHub error: " + str(errmsg))

m = YModule.FindModule(sys.argv[1]) # use serial or logical name

if m.isOnline():
    newname = sys.argv[2]
    if not YAPI.CheckLogicalName(newname):
        sys.exit("Invalid name (" + newname + ")")
    m.set_logicalName(newname)
    m.saveToFlash() # do not forget this
    print ("Module: serial= " + m.get_serialNumber()+" / name= " + m.get_logicalName())
else:
    sys.exit("not connected (check identification and USB cable)")

```

Warning: the number of write cycles of the nonvolatile memory of the module is limited. When this limit is reached, nothing guaranties that the saving process is performed correctly. This limit, linked to the technology employed by the module micro-processor, is located at about 100000 cycles. In short, you can use the `YModule.saveToFlash()` function only 100000 times in the life of the module. Make sure you do not call this function within a loop.

## Listing the modules

Obtaining the list of the connected modules is performed with the `YModule.yFirstModule()` function which returns the first module found. Then, you only need to call the `nextModule()` function of this object to find the following modules, and this as long as the returned value is not `null`. Below a short example listing the connected modules.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os,sys

from yocto_api import *

errmsg=YRefParam()

# Setup the API to use local USB devices
if YAPI.RegisterHub("usb", errmsg)!= YAPI.SUCCESS:
    sys.exit("init error"+str(errmsg))

print('Device list')

module = YModule.FirstModule()
while module is not None:
    print(module.get_serialNumber()+ ' ('+module.get_productName()+')')
    module = module.nextModule()
```

## 2.5. Error handling

When you implement a program which must interact with USB modules, you cannot disregard error handling. Inevitably, there will be a time when a user will have unplugged the device, either before running the software, or even while the software is running. The Yoctopuce library is designed to help you support this kind of behavior, but your code must nevertheless be conceived to interpret in the best possible way the errors indicated by the library.

The simplest way to work around the problem is the one used in the short examples provided in this chapter: before accessing a module, check that it is online with the `isOnline` function, and then hope that it will stay so during the fraction of a second necessary for the following code lines to run. This method is not perfect, but it can be sufficient in some cases. You must however be aware that you cannot completely exclude an error which would occur after the call to `isOnline` and which could crash the software. The only way to prevent this is to implement one of the two error handling techniques described below.

The method recommended by most programming languages for unpredictable error handling is the use of exceptions. By default, it is the behavior of the Yoctopuce library. If an error happens while you try to access a module, the library throws an exception. In this case, there are three possibilities:

- If your code catches the exception and handles it, everything goes well.
- If your program is running in debug mode, you can relatively easily determine where the problem happened and view the explanatory message linked to the exception.
- Otherwise... the exception makes your program crash, bang!

As this latest situation is not the most desirable, the Yoctopuce library offers another possibility for error handling, allowing you to create a robust program without needing to catch exceptions at every line of code. You simply need to call the `yDisableExceptions()` function to commute the library to a mode where exceptions for all the functions are systematically replaced by specific return values, which can be tested by the caller when necessary. For each function, the name of each return value in case of error is systematically documented in the library reference. The name always follows the same logic: a `get_state()` method returns a `Y_STATE_INVALID` value, a `get_currentValue` method returns a `Y_CURRENTVALUE_INVALID` value, and so on. In any case, the returned value is of the expected type and is not a null pointer which would risk crashing your program. At worst, if you display the value without testing it, it will be outside the expected bounds for the returned value. In the case of functions which do not normally return information, the return value is `YAPI_SUCCESS` if everything went well, and a different error code in case of failure.

When you work without exceptions, you can obtain an error code and an error message explaining the source of the error. You can request them from the object which returned the error, calling the `errType()` and `errMessage()` methods. Their returned values contain the same information as in the exceptions when they are active.



### **3. Reference**

## 3.1. General functions

These general functions should be used to initialize and configure the Yoctopuce library. In most cases, a simple call to function `yRegisterHub()` should be enough. The module-specific functions `yFind...()` or `yFirst...()` should then be used to retrieve an object that provides interaction with the module.

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

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

### Global functions

#### **yCheckLogicalName(name)**

Checks if a given string is valid as logical name for a module or a function.

#### **yDisableExceptions()**

Disables the use of exceptions to report runtime errors.

#### **yEnableExceptions()**

Re-enables the use of exceptions for runtime error handling.

#### **yEnableUSBHost(osContext)**

This function is used only on Android.

#### **yFreeAPI()**

Frees dynamically allocated memory blocks used by the Yoctopuce library.

#### **yGetAPIVersion()**

Returns the version identifier for the Yoctopuce library in use.

#### **yGetTickCount()**

Returns the current value of a monotone millisecond-based time counter.

#### **yHandleEvents(errmsg)**

Maintains the device-to-library communication channel.

#### **yInitAPI(mode, errmsg)**

Initializes the Yoctopuce programming library explicitly.

#### **yPreregisterHub(url, errmsg)**

Fault-tolerant alternative to `RegisterHub()`.

#### **yRegisterDeviceArrivalCallback(arrivalCallback)**

Register a callback function, to be called each time a device is plugged.

#### **yRegisterDeviceRemovalCallback(removalCallback)**

Register a callback function, to be called each time a device is unplugged.

#### **yRegisterHub(url, errmsg)**

Setup the Yoctopuce library to use modules connected on a given machine.

#### **yRegisterHubDiscoveryCallback(hubDiscoveryCallback)**

### 3. Reference

Register a callback function, to be called each time an Network Hub send an SSDP message.

#### **yRegisterLogFunction(logfun)**

Registers a log callback function.

#### **ySelectArchitecture(arch)**

Select the architecture or the library to be loaded to access to USB.

#### **ySetDelegate(object)**

(Objective-C only) Register an object that must follow the protocol YDeviceHotPlug.

#### **ySetTimeout(callback, ms\_timeout, arguments)**

Invoke the specified callback function after a given timeout.

#### **ySleep(ms\_duration, errmsg)**

Pauses the execution flow for a specified duration.

#### **yTriggerHubDiscovery(errmsg)**

Force a hub discovery, if a callback as been registered with yRegisterDeviceRemovalCallback it will be called for each net work hub that will respond to the discovery.

#### **yUnregisterHub(url)**

Setup the Yoctopuce library to no more use modules connected on a previously registered machine with RegisterHub.

#### **yUpdateDeviceList(errmsg)**

Triggers a (re)detection of connected Yoctopuce modules.

#### **yUpdateDeviceList\_async(callback, context)**

Triggers a (re)detection of connected Yoctopuce modules.



**YAPI.CheckLogicalName()****YAPI****yCheckLogicalName()****YAPI.CheckLogicalName()**

Checks if a given string is valid as logical name for a module or a function.

js	function <b>yCheckLogicalName</b> ( <b>name</b> )
nodejs	function <b>CheckLogicalName</b> ( <b>name</b> )
php	function <b>yCheckLogicalName</b> ( <b>\$name</b> )
cpp	bool <b>yCheckLogicalName</b> ( const string& <b>name</b> )
m	BOOL <b>yCheckLogicalName</b> ( NSString * <b>name</b> )
pas	function <b>yCheckLogicalName</b> ( <b>name</b> : string): boolean
vb	function <b>yCheckLogicalName</b> ( ByVal <b>name</b> As String) As Boolean
cs	bool <b>CheckLogicalName</b> ( string <b>name</b> )
java	boolean <b>CheckLogicalName</b> ( String <b>name</b> )
py	def <b>CheckLogicalName</b> ( <b>name</b> )

A valid logical name has a maximum of 19 characters, all among A . . Z, a . . z, 0 . . 9, \_, and -. If you try to configure a logical name with an incorrect string, the invalid characters are ignored.

**Parameters :**

**name** a string containing the name to check.

**Returns :**

true if the name is valid, false otherwise.

**YAPI.DisableExceptions()****YAPI****yDisableExceptions()YAPI.DisableExceptions()**

---

Disables the use of exceptions to report runtime errors.

js	function <b>yDisableExceptions</b> ( )
nodejs	function <b>DisableExceptions</b> ( )
php	function <b>yDisableExceptions</b> ( )
cpp	void <b>yDisableExceptions</b> ( )
m	void <b>yDisableExceptions</b> ( )
pas	procedure <b>yDisableExceptions</b> ( )
vb	procedure <b>yDisableExceptions</b> ( )
cs	void <b>DisableExceptions</b> ( )
py	def <b>DisableExceptions</b> ( )

When exceptions are disabled, every function returns a specific error value which depends on its type and which is documented in this reference manual.

**YAPI.EnableExceptions()****YAPI****yEnableExceptions()YAPI.EnableExceptions()**

Re-enables the use of exceptions for runtime error handling.

js	function <b>yEnableExceptions</b> ( )
nodejs	function <b>EnableExceptions</b> ( )
php	function <b>yEnableExceptions</b> ( )
cpp	void <b>yEnableExceptions</b> ( )
m	void <b>yEnableExceptions</b> ( )
pas	procedure <b>yEnableExceptions</b> ( )
vb	procedure <b>yEnableExceptions</b> ( )
cs	void <b>EnableExceptions</b> ( )
py	def <b>EnableExceptions</b> ( )

Be aware than when exceptions are enabled, every function that fails triggers an exception. If the exception is not caught by the user code, it either fires the debugger or aborts (i.e. crash) the program. On failure, throws an exception or returns a negative error code.

## YAPI.EnableUSBHost() yEnableUSBHost()

YAPI

This function is used only on Android.

```
java void EnableUSBHost( Object osContext)
```

Before calling `yRegisterHub( "usb" )` you need to activate the USB host port of the system. This function takes as argument, an object of class `android.content.Context` (or any subclass). It is not necessary to call this function to reach modules through the network.

### Parameters :

**osContext** an object of class `android.content.Context` (or any subclass).

**YAPI.FreeAPI()****YAPI****yFreeAPI()****YAPI.FreeAPI()**

Frees dynamically allocated memory blocks used by the Yoctopuce library.

js	function <b>yFreeAPI</b> ( )
nodejs	function <b>FreeAPI</b> ( )
php	function <b>yFreeAPI</b> ( )
cpp	void <b>yFreeAPI</b> ( )
m	void <b>yFreeAPI</b> ( )
pas	procedure <b>yFreeAPI</b> ( )
vb	procedure <b>yFreeAPI</b> ( )
cs	void <b>FreeAPI</b> ( )
java	void <b>FreeAPI</b> ( )
py	def <b>FreeAPI</b> ( )

It is generally not required to call this function, unless you want to free all dynamically allocated memory blocks in order to track a memory leak for instance. You should not call any other library function after calling `yFreeAPI( )`, or your program will crash.

**YAPI.GetAPIVersion()****YAPI****yGetAPIVersion()****YAPI.GetAPIVersion()**

Returns the version identifier for the Yoctopuce library in use.

js	function <b>yGetAPIVersion</b> ( )
nodejs	function <b>GetAPIVersion</b> ( )
php	function <b>yGetAPIVersion</b> ( )
cpp	string <b>yGetAPIVersion</b> ( )
m	NSString* <b>yGetAPIVersion</b> ( )
pas	function <b>yGetAPIVersion</b> ( ): string
vb	function <b>yGetAPIVersion</b> ( ) As String
cs	String <b>GetAPIVersion</b> ( )
java	String <b>GetAPIVersion</b> ( )
py	def <b>GetAPIVersion</b> ( )

The version is a string in the form "Major.Minor.Build", for instance "1.01.5535". For languages using an external DLL (for instance C#, VisualBasic or Delphi), the character string includes as well the DLL version, for instance "1.01.5535 (1.01.5439)".

If you want to verify in your code that the library version is compatible with the version that you have used during development, verify that the major number is strictly equal and that the minor number is greater or equal. The build number is not relevant with respect to the library compatibility.

**Returns :**

a character string describing the library version.

**YAPI.GetTickCount()****YAPI****yGetTickCount()****YAPI.GetTickCount()**

Returns the current value of a monotone millisecond-based time counter.

js	function <b>yGetTickCount</b> ( )
nodejs	function <b>GetTickCount</b> ( )
php	function <b>yGetTickCount</b> ( )
cpp	u64 <b>yGetTickCount</b> ( )
m	u64 <b>yGetTickCount</b> ( )
pas	function <b>yGetTickCount</b> ( ): u64
vb	function <b>yGetTickCount</b> ( ) As Long
cs	ulong <b>GetTickCount</b> ( )
java	long <b>GetTickCount</b> ( )
py	def <b>GetTickCount</b> ( )

This counter can be used to compute delays in relation with Yoctopuce devices, which also uses the millisecond as timebase.

**Returns :**

a long integer corresponding to the millisecond counter.

## YAPI.HandleEvents()

YAPI

### yHandleEvents()YAPI.HandleEvents()

Maintains the device-to-library communication channel.

js	function <b>yHandleEvents</b> ( <b>errmsg</b> )
nodejs	function <b>HandleEvents</b> ( <b>errmsg</b> )
php	function <b>yHandleEvents</b> ( <b>&amp;\$errmsg</b> )
cpp	YRETCODE <b>yHandleEvents</b> ( string& <b>errmsg</b> )
m	YRETCODE <b>yHandleEvents</b> ( NSError** <b>errmsg</b> )
pas	function <b>yHandleEvents</b> ( var <b>errmsg</b> : string): integer
vb	function <b>yHandleEvents</b> ( ByRef <b>errmsg</b> As String) As YRETCODE
cs	YRETCODE <b>HandleEvents</b> ( ref string <b>errmsg</b> )
java	int <b>HandleEvents</b> ( )
py	def <b>HandleEvents</b> ( <b>errmsg</b> =None)

If your program includes significant loops, you may want to include a call to this function to make sure that the library takes care of the information pushed by the modules on the communication channels. This is not strictly necessary, but it may improve the reactivity of the library for the following commands.

This function may signal an error in case there is a communication problem while contacting a module.

#### Parameters :

**errmsg** a string passed by reference to receive any error message.

#### Returns :

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.



**YAPI.InitAPI()****YAPI****yInitAPI()YAPI.InitAPI()**

Initializes the Yoctopuce programming library explicitly.

js	function <b>yInitAPI</b> ( <b>mode</b> , <b>errmsg</b> )
nodejs	function <b>InitAPI</b> ( <b>mode</b> , <b>errmsg</b> )
php	function <b>yInitAPI</b> ( <b>\$mode</b> , <b>&amp;\$errmsg</b> )
cpp	YRETCODE <b>yInitAPI</b> ( int <b>mode</b> , string& <b>errmsg</b> )
m	YRETCODE <b>yInitAPI</b> ( int <b>mode</b> , NSError** <b>errmsg</b> )
pas	function <b>yInitAPI</b> ( <b>mode</b> : integer, var <b>errmsg</b> : string): integer
vb	function <b>yInitAPI</b> ( ByVal <b>mode</b> As Integer, ByRef <b>errmsg</b> As String) As Integer
cs	int <b>InitAPI</b> ( int <b>mode</b> , ref string <b>errmsg</b> )
java	int <b>InitAPI</b> ( int <b>mode</b> )
py	def <b>InitAPI</b> ( <b>mode</b> , <b>errmsg=None</b> )

It is not strictly needed to call `yInitAPI()`, as the library is automatically initialized when calling `yRegisterHub()` for the first time.

When `Y_DETECT_NONE` is used as detection mode, you must explicitly use `yRegisterHub()` to point the API to the VirtualHub on which your devices are connected before trying to access them.

**Parameters :**

**mode** an integer corresponding to the type of automatic device detection to use. Possible values are `Y_DETECT_NONE`, `Y_DETECT_USB`, `Y_DETECT_NET`, and `Y_DETECT_ALL`.

**errmsg** a string passed by reference to receive any error message.

**Returns :**

`YAPI_SUCCESS` when the call succeeds. On failure, throws an exception or returns a negative error code.

**YAPI.PreregisterHub()****yPreregisterHub()YAPI.PreregisterHub()**

Fault-tolerant alternative to RegisterHub().

js	function <b>yPreregisterHub</b> ( <b>url</b> , <b>errmsg</b> )
nodejs	function <b>PreregisterHub</b> ( <b>url</b> , <b>errmsg</b> )
php	function <b>yPreregisterHub</b> ( <b>\$url</b> , <b>&amp;\$errmsg</b> )
cpp	YRETCODE <b>yPreregisterHub</b> ( const string& <b>url</b> , string& <b>errmsg</b> )
m	YRETCODE <b>yPreregisterHub</b> ( NSString * <b>url</b> , NSError** <b>errmsg</b> )
pas	function <b>yPreregisterHub</b> ( <b>url</b> : string, var <b>errmsg</b> : string): integer
vb	function <b>yPreregisterHub</b> ( ByVal <b>url</b> As String, ByRef <b>errmsg</b> As String) As Integer
cs	int <b>PreregisterHub</b> ( string <b>url</b> , ref string <b>errmsg</b> )
java	int <b>PreregisterHub</b> ( String <b>url</b> )
py	def <b>PreregisterHub</b> ( <b>url</b> , <b>errmsg=None</b> )

This function has the same purpose and same arguments as RegisterHub( ), but does not trigger an error when the selected hub is not available at the time of the function call. This makes it possible to register a network hub independently of the current connectivity, and to try to contact it only when a device is actively needed.

**Parameters :**

**url** a string containing either "usb", "callback" or the root URL of the hub to monitor  
**errmsg** a string passed by reference to receive any error message.

**Returns :**

YAPI\_SUCCESS when the call succeeds.

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

## YAPI.RegisterDeviceArrivalCallback()

## yRegisterDeviceArrivalCallback()

## YAPI.RegisterDeviceArrivalCallback()

YAPI

Register a callback function, to be called each time a device is plugged.

js	function <b>yRegisterDeviceArrivalCallback</b> ( <b>arrivalCallback</b> )
nodejs	function <b>RegisterDeviceArrivalCallback</b> ( <b>arrivalCallback</b> )
php	function <b>yRegisterDeviceArrivalCallback</b> ( <b>\$arrivalCallback</b> )
cpp	void <b>yRegisterDeviceArrivalCallback</b> ( yDeviceUpdateCallback <b>arrivalCallback</b> )
m	void <b>yRegisterDeviceArrivalCallback</b> ( yDeviceUpdateCallback <b>arrivalCallback</b> )
pas	procedure <b>yRegisterDeviceArrivalCallback</b> ( <b>arrivalCallback</b> : yDeviceUpdateFunc)
vb	procedure <b>yRegisterDeviceArrivalCallback</b> ( ByVal <b>arrivalCallback</b> As yDeviceUpdateFunc)
cs	void <b>RegisterDeviceArrivalCallback</b> ( yDeviceUpdateFunc <b>arrivalCallback</b> )
java	void <b>RegisterDeviceArrivalCallback</b> ( DeviceArrivalCallback <b>arrivalCallback</b> )
py	def <b>RegisterDeviceArrivalCallback</b> ( <b>arrivalCallback</b> )

This callback will be invoked while `yUpdateDeviceList` is running. You will have to call this function on a regular basis.

### Parameters :

**arrivalCallback** a procedure taking a `YModule` parameter, or null

**YAPI.RegisterDeviceRemovalCallback()****yRegisterDeviceRemovalCallback()****YAPI.RegisterDeviceRemovalCallback()**

Register a callback function, to be called each time a device is unplugged.

js	function <b>yRegisterDeviceRemovalCallback</b> ( <b>removalCallback</b> )
nodejs	function <b>RegisterDeviceRemovalCallback</b> ( <b>removalCallback</b> )
php	function <b>yRegisterDeviceRemovalCallback</b> ( <b>\$removalCallback</b> )
cpp	void <b>yRegisterDeviceRemovalCallback</b> ( yDeviceUpdateCallback <b>removalCallback</b> )
m	void <b>yRegisterDeviceRemovalCallback</b> ( yDeviceUpdateCallback <b>removalCallback</b> )
pas	procedure <b>yRegisterDeviceRemovalCallback</b> ( <b>removalCallback</b> : yDeviceUpdateFunc)
vb	procedure <b>yRegisterDeviceRemovalCallback</b> ( ByVal <b>removalCallback</b> As yDeviceUpdateFunc)
cs	void <b>RegisterDeviceRemovalCallback</b> ( yDeviceUpdateFunc <b>removalCallback</b> )
java	void <b>RegisterDeviceRemovalCallback</b> ( DeviceRemovalCallback <b>removalCallback</b> )
py	def <b>RegisterDeviceRemovalCallback</b> ( <b>removalCallback</b> )

This callback will be invoked while `yUpdateDeviceList` is running. You will have to call this function on a regular basis.

**Parameters :**

**removalCallback** a procedure taking a `YModule` parameter, or `null`

## YAPI.RegisterHub() yRegisterHub()YAPI.RegisterHub()

YAPI

Setup the Yoctopuce library to use modules connected on a given machine.

js	function <b>yRegisterHub</b> ( <b>url</b> , <b>errmsg</b> )
nodejs	function <b>RegisterHub</b> ( <b>url</b> , <b>errmsg</b> )
php	function <b>yRegisterHub</b> ( <b>\$url</b> , <b>&amp;\$errmsg</b> )
cpp	YRETCODE <b>yRegisterHub</b> ( const string& <b>url</b> , string& <b>errmsg</b> )
m	YRETCODE <b>yRegisterHub</b> ( NSString * <b>url</b> , NSError** <b>errmsg</b> )
pas	function <b>yRegisterHub</b> ( <b>url</b> : string, var <b>errmsg</b> : string): integer
vb	function <b>yRegisterHub</b> ( ByVal <b>url</b> As String, ByRef <b>errmsg</b> As String) As Integer
cs	int <b>RegisterHub</b> ( string <b>url</b> , ref string <b>errmsg</b> )
java	int <b>RegisterHub</b> ( String <b>url</b> )
py	def <b>RegisterHub</b> ( <b>url</b> , <b>errmsg</b> =None)

The parameter will determine how the API will work. Use the following values:

**usb**: When the **usb** keyword is used, the API will work with devices connected directly to the USB bus. Some programming languages such as Javascript, PHP, and Java don't provide direct access to USB hardware, so **usb** will not work with these. In this case, use a VirtualHub or a networked YoctoHub (see below).

**x.x.x.x** or **hostname**: The API will use the devices connected to the host with the given IP address or hostname. That host can be a regular computer running a VirtualHub, or a networked YoctoHub such as YoctoHub-Ethernet or YoctoHub-Wireless. If you want to use the VirtualHub running on your local computer, use the IP address 127.0.0.1.

**callback**: that keyword makes the API run in "*HTTP Callback*" mode. This is a special mode allowing to take control of Yoctopuce devices through a NAT filter when using a VirtualHub or a networked YoctoHub. You only need to configure your hub to call your server script on a regular basis. This mode is currently available for PHP and Node.JS only.

Be aware that only one application can use direct USB access at a given time on a machine. Multiple access would cause conflicts while trying to access the USB modules. In particular, this means that you must stop the VirtualHub software before starting an application that uses direct USB access. The workaround for this limitation is to setup the library to use the VirtualHub rather than direct USB access.

If access control has been activated on the hub, virtual or not, you want to reach, the URL parameter should look like:

```
http://username:password@adresse:port
```

You can call *RegisterHub* several times to connect to several machines.

### Parameters :

**url** a string containing either "**usb**", "**callback**" or the root URL of the hub to monitor  
**errmsg** a string passed by reference to receive any error message.

### Returns :

YAPI\_SUCCESS when the call succeeds.

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

## YAPI.RegisterHubDiscoveryCallback() yRegisterHubDiscoveryCallback() YAPI.RegisterHubDiscoveryCallback()

YAPI

Register a callback function, to be called each time an Network Hub send an SSDP message.

cpp	void <b>yRegisterHubDiscoveryCallback</b> ( YHubDiscoveryCallback <b>hubDiscoveryCallback</b> )
m	+(void) <b>yRegisterHubDiscoveryCallback</b> : (YHubDiscoveryCallback) <b>hubDiscoveryCallback</b>
pas	procedure <b>yRegisterHubDiscoveryCallback</b> ( <b>hubDiscoveryCallback</b> : YHubDiscoveryCallback)
vb	procedure <b>yRegisterHubDiscoveryCallback</b> ( ByVal <b>hubDiscoveryCallback</b> As YHubDiscoveryCallback)
cs	void <b>RegisterHubDiscoveryCallback</b> ( YHubDiscoveryCallback <b>hubDiscoveryCallback</b> )
java	void <b>RegisterHubDiscoveryCallback</b> ( HubDiscoveryCallback <b>hubDiscoveryCallback</b> )
py	def <b>RegisterHubDiscoveryCallback</b> ( <b>hubDiscoveryCallback</b> )

The callback has two string parameter, the first one contain the serial number of the hub and the second contain the URL of the network hub (this URL can be passed to RegisterHub). This callback will be invoked while yUpdateDeviceList is running. You will have to call this function on a regular basis.

### Parameters :

**hubDiscoveryCallback** a procedure taking two string parameter, or null

**YAPI.RegisterLogFunction()****YAPI****yRegisterLogFunction()YAPI.RegisterLogFunction()**

Registers a log callback function.

cpp	void <b>yRegisterLogFunction</b> ( yLogFunction <b>logfun</b> )
m	void <b>yRegisterLogFunction</b> ( yLogCallback <b>logfun</b> )
pas	procedure <b>yRegisterLogFunction</b> ( <b>logfun</b> : yLogFunc)
vb	procedure <b>yRegisterLogFunction</b> ( ByVal <b>logfun</b> As yLogFunc)
cs	void <b>RegisterLogFunction</b> ( yLogFunc <b>logfun</b> )
java	void <b>RegisterLogFunction</b> ( LogCallback <b>logfun</b> )
py	def <b>RegisterLogFunction</b> ( <b>logfun</b> )

This callback will be called each time the API have something to say. Quite useful to debug the API.

**Parameters :**

**logfun** a procedure taking a string parameter, or null

**YAPI.SelectArchitecture()****YAPI****ySelectArchitecture()YAPI.SelectArchitecture()**

Select the architecture or the library to be loaded to access to USB.

```
py def SelectArchitecture( arch)
```

By default, the Python library automatically detects the appropriate library to use. However, for Linux ARM, it not possible to reliably distinguish between a Hard Float (armhf) and a Soft Float (armel) install. For in this case, it is therefore recommended to manually select the proper architecture by calling `SelectArchitecture()` before any other call to the library.

**Parameters :**

**arch** A string containing the architecture to use. Possibles value are: "armhf","armel", "i386","x86\_64","32bit", "64bit"

**Returns :**

nothing.

On failure, throws an exception.



## YAPI.SetDelegate() ySetDelegate()

## YAPI

(Objective-C only) Register an object that must follow the protocol YDeviceHotPlug.

```
m void ySetDelegate( id object)
```

The methods `yDeviceArrival` and `yDeviceRemoval` will be invoked while `yUpdateDeviceList` is running. You will have to call this function on a regular basis.

### Parameters :

**object** an object that must follow the protocol YAPIDelegate, or nil

## YAPI.SetTimeout() ySetTimeout()

YAPI

Invoke the specified callback function after a given timeout.

```
js function ySetTimeout( callback, ms_timeout, arguments)
nodejs function SetTimeout( callback, ms_timeout, arguments)
```

This function behaves more or less like Javascript `setTimeout`, but during the waiting time, it will call `yHandleEvents` and `yUpdateDeviceList` periodically, in order to keep the API up-to-date with current devices.

### Parameters :

- callback** the function to call after the timeout occurs. On Microsoft Internet Explorer, the callback must be provided as a string to be evaluated.
- ms\_timeout** an integer corresponding to the duration of the timeout, in milliseconds.
- arguments** additional arguments to be passed to the callback function can be provided, if needed (not supported on Microsoft Internet Explorer).

### Returns :

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**YAPI.Sleep()****YAPI****ySleep()YAPI.Sleep()**

Pauses the execution flow for a specified duration.

js	function <b>ySleep</b> ( <b>ms_duration</b> , <b>errmsg</b> )
nodejs	function <b>Sleep</b> ( <b>ms_duration</b> , <b>errmsg</b> )
php	function <b>ySleep</b> ( <b>\$ms_duration</b> , <b>&amp;\$errmsg</b> )
cpp	YRETCODE <b>ySleep</b> ( unsigned <b>ms_duration</b> , string& <b>errmsg</b> )
m	YRETCODE <b>ySleep</b> ( unsigned <b>ms_duration</b> , NSError ** <b>errmsg</b> )
pas	function <b>ySleep</b> ( <b>ms_duration</b> : integer, var <b>errmsg</b> : string): integer
vb	function <b>ySleep</b> ( ByVal <b>ms_duration</b> As Integer, ByRef <b>errmsg</b> As String) As Integer
cs	int <b>Sleep</b> ( int <b>ms_duration</b> , ref string <b>errmsg</b> )
java	int <b>Sleep</b> ( long <b>ms_duration</b> )
py	def <b>Sleep</b> ( <b>ms_duration</b> , <b>errmsg=None</b> )

This function implements a passive waiting loop, meaning that it does not consume CPU cycles significantly. The processor is left available for other threads and processes. During the pause, the library nevertheless reads from time to time information from the Yoctopuce modules by calling `yHandleEvents()`, in order to stay up-to-date.

This function may signal an error in case there is a communication problem while contacting a module.

**Parameters :**

**ms\_duration** an integer corresponding to the duration of the pause, in milliseconds.  
**errmsg** a string passed by reference to receive any error message.

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**YAPI.TriggerHubDiscovery()****YAPI****yTriggerHubDiscovery()** **YAPI.TriggerHubDiscovery()**

Force a hub discovery, if a callback as been registered with `yRegisterDeviceRemovalCallback` it will be called for each net work hub that will respond to the discovery.

cpp	<code>YRETCODE yTriggerHubDiscovery( string&amp; errmsg)</code>
m	<code>+(YRETCODE) yTriggerHubDiscovery : (NSError**) errmsg</code>
pas	<code>function yTriggerHubDiscovery( var errmsg: string): integer</code>
vb	<code>function yTriggerHubDiscovery( ByRef errmsg As String) As Integer</code>
cs	<code>int TriggerHubDiscovery( ref string errmsg)</code>
java	<code>int TriggerHubDiscovery( )</code>
py	<code>def TriggerHubDiscovery( errmsg=None)</code>

**Parameters :**

**errmsg** a string passed by reference to receive any error message.

**Returns :**

`YAPI_SUCCESS` when the call succeeds. On failure, throws an exception or returns a negative error code.

**YAPI.UnregisterHub()****YAPI****yUnregisterHub()****YAPI.UnregisterHub()**

Setup the Yoctopuce library to no more use modules connected on a previously registered machine with RegisterHub.

js	function <b>yUnregisterHub</b> ( <b>url</b> )
nodejs	function <b>UnregisterHub</b> ( <b>url</b> )
php	function <b>yUnregisterHub</b> ( <b>\$url</b> )
cpp	void <b>yUnregisterHub</b> ( const string& <b>url</b> )
m	void <b>yUnregisterHub</b> ( NSString * <b>url</b> )
pas	procedure <b>yUnregisterHub</b> ( <b>url</b> : string)
vb	procedure <b>yUnregisterHub</b> ( ByVal <b>url</b> As String)
cs	void <b>UnregisterHub</b> ( string <b>url</b> )
java	void <b>UnregisterHub</b> ( String <b>url</b> )
py	def <b>UnregisterHub</b> ( <b>url</b> )

**Parameters :**

**url** a string containing either "usb" or the

**YAPI.UpdateDeviceList()****YAPI****yUpdateDeviceList()****YAPI.UpdateDeviceList()**

Triggers a (re)detection of connected Yoctopuce modules.

js	function <b>yUpdateDeviceList</b> ( <b>errmsg</b> )
nodejs	function <b>UpdateDeviceList</b> ( <b>errmsg</b> )
php	function <b>yUpdateDeviceList</b> ( <b>&amp;\$errmsg</b> )
cpp	YRETCODE <b>yUpdateDeviceList</b> ( string& <b>errmsg</b> )
m	YRETCODE <b>yUpdateDeviceList</b> ( NSError** <b>errmsg</b> )
pas	function <b>yUpdateDeviceList</b> ( var <b>errmsg</b> : string): integer
vb	function <b>yUpdateDeviceList</b> ( ByRef <b>errmsg</b> As String) As YRETCODE
cs	YRETCODE <b>UpdateDeviceList</b> ( ref string <b>errmsg</b> )
java	int <b>UpdateDeviceList</b> ( )
py	def <b>UpdateDeviceList</b> ( <b>errmsg</b> =None)

The library searches the machines or USB ports previously registered using `yRegisterHub()`, and invokes any user-defined callback function in case a change in the list of connected devices is detected.

This function can be called as frequently as desired to refresh the device list and to make the application aware of hot-plug events.

**Parameters :**

**errmsg** a string passed by reference to receive any error message.

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

## YAPI.UpdateDeviceList\_async() yUpdateDeviceList\_async()

YAPI

Triggers a (re)detection of connected Yoctopuce modules.

```
js function yUpdateDeviceList_async( callback, context)
nodejs function UpdateDeviceList_async( callback, context)
```

The library searches the machines or USB ports previously registered using `yRegisterHub()`, and invokes any user-defined callback function in case a change in the list of connected devices is detected.

This function can be called as frequently as desired to refresh the device list and to make the application aware of hot-plug events.

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.

### Parameters :

- callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the result code (`YAPI_SUCCESS` if the operation completes successfully) and the error message.
- context** caller-specific object that is passed as-is to the callback function

### Returns :

nothing : the result is provided to the callback.

## 3.2. Accelerometer function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindAccelerometer(func)

Retrieves an accelerometer for a given identifier.

#### yFirstAccelerometer()

Starts the enumeration of accelerometers currently accessible.

### YAccelerometer methods

#### accelerometer→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### accelerometer→describe()

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

#### accelerometer→get\_advertisedValue()

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

#### accelerometer→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### accelerometer→get\_currentValue()

Returns the current value of the acceleration.

#### accelerometer→get\_errorMessage()

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

#### accelerometer→get\_errorType()

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

#### accelerometer→get\_friendlyName()

Returns a global identifier of the accelerometer in the format MODULE\_NAME . FUNCTION\_NAME.

#### accelerometer→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### accelerometer→get\_functionId()

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

#### accelerometer→get\_hardwareId()

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



**accelerometer→get\_highestValue()**

Returns the maximal value observed for the acceleration since the device was started.

**accelerometer→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**accelerometer→get\_logicalName()**

Returns the logical name of the accelerometer.

**accelerometer→get\_lowestValue()**

Returns the minimal value observed for the acceleration since the device was started.

**accelerometer→get\_module()**

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

**accelerometer→get\_module\_async(callback, context)**

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

**accelerometer→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**accelerometer→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**accelerometer→get\_resolution()**

Returns the resolution of the measured values.

**accelerometer→get\_unit()**

Returns the measuring unit for the acceleration.

**accelerometer→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**accelerometer→get\_xValue()**

Returns the X component of the acceleration, as a floating point number.

**accelerometer→get\_yValue()**

Returns the Y component of the acceleration, as a floating point number.

**accelerometer→get\_zValue()**

Returns the Z component of the acceleration, as a floating point number.

**accelerometer→isOnline()**

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

**accelerometer→isOnline\_async(callback, context)**

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

**accelerometer→load(msValidity)**

Preloads the accelerometer cache with a specified validity duration.

**accelerometer→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**accelerometer→load\_async(msValidity, callback, context)**

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

**accelerometer→nextAccelerometer()**

Continues the enumeration of accelerometers started using yFirstAccelerometer().

**accelerometer→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**accelerometer→registerValueCallback(callback)**

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

### 3. Reference

#### **accelerometer→set\_highestValue(newval)**

Changes the recorded maximal value observed.

#### **accelerometer→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

#### **accelerometer→set\_logicalName(newval)**

Changes the logical name of the accelerometer.

#### **accelerometer→set\_lowestValue(newval)**

Changes the recorded minimal value observed.

#### **accelerometer→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

#### **accelerometer→set\_resolution(newval)**

Changes the resolution of the measured physical values.

#### **accelerometer→set\_userData(data)**

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

#### **accelerometer→wait\_async(callback, context)**

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

## YAccelerometer.FindAccelerometer() yFindAccelerometer() YAccelerometer.FindAccelerometer()

## YAccelerometer

Retrieves an accelerometer for a given identifier.

js	function <b>yFindAccelerometer</b> ( <b>func</b> )
nodejs	function <b>FindAccelerometer</b> ( <b>func</b> )
php	function <b>yFindAccelerometer</b> ( <b>\$func</b> )
cpp	YAccelerometer* <b>yFindAccelerometer</b> ( const string& <b>func</b> )
m	YAccelerometer* <b>yFindAccelerometer</b> ( NSString* <b>func</b> )
pas	function <b>yFindAccelerometer</b> ( <b>func</b> : string): TYAccelerometer
vb	function <b>yFindAccelerometer</b> ( ByVal <b>func</b> As String) As YAccelerometer
cs	YAccelerometer <b>FindAccelerometer</b> ( string <b>func</b> )
java	YAccelerometer <b>FindAccelerometer</b> ( String <b>func</b> )
py	def <b>FindAccelerometer</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the accelerometer is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YAccelerometer.isOnline()` to test if the accelerometer is indeed online at a given time. In case of ambiguity when looking for an accelerometer 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 accelerometer

### Returns :

a `YAccelerometer` object allowing you to drive the accelerometer.

## YAccelerometer.FirstAccelerometer() yFirstAccelerometer() YAccelerometer.FirstAccelerometer()

YAccelerometer

Starts the enumeration of accelerometers currently accessible.

js	function <b>yFirstAccelerometer</b> ( )
nodejs	function <b>FirstAccelerometer</b> ( )
php	function <b>yFirstAccelerometer</b> ( )
cpp	YAccelerometer* <b>yFirstAccelerometer</b> ( )
m	YAccelerometer* <b>yFirstAccelerometer</b> ( )
pas	function <b>yFirstAccelerometer</b> ( ): TYAccelerometer
vb	function <b>yFirstAccelerometer</b> ( ) As YAccelerometer
cs	YAccelerometer <b>FirstAccelerometer</b> ( )
java	YAccelerometer <b>FirstAccelerometer</b> ( )
py	def <b>FirstAccelerometer</b> ( )

Use the method `YAccelerometer.nextAccelerometer( )` to iterate on next accelerometers.

### Returns :

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

## accelerometer→calibrateFromPoints() accelerometer.calibrateFromPoints()

## YAccelerometer

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

js	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
nodejs	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
php	function <b>calibrateFromPoints</b> ( <b>\$rawValues</b> , <b>\$refValues</b> )
cpp	int <b>calibrateFromPoints</b> ( vector<double> <b>rawValues</b> , vector<double> <b>refValues</b> )
m	-(int) <b>calibrateFromPoints</b> : (NSMutableArray*) <b>rawValues</b> : (NSMutableArray*) <b>refValues</b>
pas	function <b>calibrateFromPoints</b> ( <b>rawValues</b> : TDoubleArray, <b>refValues</b> : TDoubleArray): LongInt
vb	procedure <b>calibrateFromPoints</b> ( )
cs	int <b>calibrateFromPoints</b> ( List<double> <b>rawValues</b> , List<double> <b>refValues</b> )
java	int <b>calibrateFromPoints</b> ( ArrayList<Double> <b>rawValues</b> , ArrayList<Double> <b>refValues</b> )
py	def <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
cmd	YAccelerometer <b>target</b> <b>calibrateFromPoints</b> <b>rawValues</b> <b>refValues</b>

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→describe()****accelerometer.describe()****YAccelerometer**

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

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

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 accelerometer (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**accelerometer→get\_advertisedValue()****YAccelerometer****accelerometer→advertisedValue()****accelerometer.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the accelerometer (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**accelerometer**→**get\_currentRawValue()****YAccelerometer****accelerometer**→**currentRawValue()****accelerometer.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.



**accelerometer→get\_currentValue()****YAccelerometer****accelerometer→currentValue()****accelerometer.get\_currentValue()**

Returns the current value of the acceleration.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the acceleration

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**accelerometer→get\_errorMessage()****YAccelerometer****accelerometer→errorMessage()****accelerometer.get\_errorMessage()**

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

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the accelerometer object

**accelerometer→get\_errorType()****YAccelerometer****accelerometer→errorType()****accelerometer.get\_errorType()**

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

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the accelerometer object

**accelerometer→get\_friendlyName()**  
**accelerometer→friendlyName()**  
**accelerometer.get\_friendlyName()**

**YAccelerometer**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

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

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

**accelerometer→get\_functionDescriptor()****YAccelerometer****accelerometer→functionDescriptor()****accelerometer.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**accelerometer→get\_functionId()**  
**accelerometer→functionId()**  
**accelerometer.get\_functionId()**

**YAccelerometer**

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

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

For example `relay1`

**Returns :**

a string that identifies the accelerometer (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**accelerometer→get\_hardwareId()****YAccelerometer****accelerometer→hardwareId()****accelerometer.get\_hardwareId()**

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

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

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

**Returns :**

a string that uniquely identifies the accelerometer (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**accelerometer→get\_highestValue()****YAccelerometer****accelerometer→highestValue()****accelerometer.get\_highestValue()**

Returns the maximal value observed for the acceleration since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the acceleration since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.



**accelerometer→get\_logFrequency()****YAccelerometer****accelerometer→logFrequency()****accelerometer.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**accelerometer→get\_logicalName()**  
**accelerometer→logicalName()**  
**accelerometer.get\_logicalName()**

**YAccelerometer**

Returns the logical name of the accelerometer.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the accelerometer. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**accelerometer→get\_lowestValue()****YAccelerometer****accelerometer→lowestValue()****accelerometer.get\_lowestValue()**

Returns the minimal value observed for the acceleration since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the acceleration since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**accelerometer**→**get\_module()****YAccelerometer****accelerometer**→**module()****accelerometer.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**accelerometer→get\_module\_async()****YAccelerometer****accelerometer→module\_async()**

---

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**accelerometer→get\_recordedData()****YAccelerometer****accelerometer→recordedData()****accelerometer.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
c++	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YAccelerometer <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**accelerometer→get\_reportFrequency()****YAccelerometer****accelerometer→reportFrequency()****accelerometer.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**accelerometer**→**get\_resolution()****YAccelerometer****accelerometer**→**resolution()****accelerometer.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.



**accelerometer**→**get\_unit()****YAccelerometer****accelerometer**→**unit()****accelerometer.get\_unit()**

Returns the measuring unit for the acceleration.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the acceleration

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**accelerometer→get\_userData()****YAccelerometer****accelerometer→userData()****accelerometer.get\_userData()**

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

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): TObject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**accelerometer→get\_xValue()****YAccelerometer****accelerometer→xValue()accelerometer.get\_xValue()**

Returns the X component of the acceleration, as a floating point number.

js	function <b>get_xValue</b> ( )
nodejs	function <b>get_xValue</b> ( )
php	function <b>get_xValue</b> ( )
cpp	double <b>get_xValue</b> ( )
m	-(double) xValue
pas	function <b>get_xValue</b> ( ): double
vb	function <b>get_xValue</b> ( ) As Double
cs	double <b>get_xValue</b> ( )
java	double <b>get_xValue</b> ( )
py	def <b>get_xValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_xValue</b>

**Returns :**

a floating point number corresponding to the X component of the acceleration, as a floating point number

On failure, throws an exception or returns Y\_XVALUE\_INVALID.

**accelerometer**→**get\_yValue()****YAccelerometer****accelerometer**→**yValue()****accelerometer.get\_yValue()**

Returns the Y component of the acceleration, as a floating point number.

js	function <b>get_yValue</b> ( )
nodejs	function <b>get_yValue</b> ( )
php	function <b>get_yValue</b> ( )
cpp	double <b>get_yValue</b> ( )
m	-(double) yValue
pas	function <b>get_yValue</b> ( ): double
vb	function <b>get_yValue</b> ( ) As Double
cs	double <b>get_yValue</b> ( )
java	double <b>get_yValue</b> ( )
py	def <b>get_yValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_yValue</b>

**Returns :**

a floating point number corresponding to the Y component of the acceleration, as a floating point number

On failure, throws an exception or returns Y\_YVALUE\_INVALID.

**accelerometer→get\_zValue()****YAccelerometer****accelerometer→zValue()accelerometer.get\_zValue()**

Returns the Z component of the acceleration, as a floating point number.

js	function <b>get_zValue</b> ( )
nodejs	function <b>get_zValue</b> ( )
php	function <b>get_zValue</b> ( )
cpp	double <b>get_zValue</b> ( )
m	-(double) zValue
pas	function <b>get_zValue</b> ( ): double
vb	function <b>get_zValue</b> ( ) As Double
cs	double <b>get_zValue</b> ( )
java	double <b>get_zValue</b> ( )
py	def <b>get_zValue</b> ( )
cmd	YAccelerometer <b>target</b> <b>get_zValue</b>

**Returns :**

a floating point number corresponding to the Z component of the acceleration, as a floating point number

On failure, throws an exception or returns Y\_ZVALUE\_INVALID.

**accelerometer**→**isOnline()****accelerometer.isOnline()****YAccelerometer**

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

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

If there is a cached value for the accelerometer 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 accelerometer.

**Returns :**

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

**accelerometer→isOnline\_async()****YAccelerometer**

Checks if the accelerometer 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 accelerometer 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.

**accelerometer**→**load()****accelerometer.load()****YAccelerometer**

Preloads the accelerometer cache with a specified validity duration.

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

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 traffic 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.



## accelerometer→loadCalibrationPoints() accelerometer.loadCalibrationPoints()

## YAccelerometer

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js      function loadCalibrationPoints( rawValues, refValues)
nodejs  function loadCalibrationPoints( rawValues, refValues)
php     function loadCalibrationPoints( &$rawValues, &$refValues)
cpp     int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m       -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )
cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)
cmd     YAccelerometer target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→load\_async()****YAccelerometer**

Preloads the accelerometer 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 traffic 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.

## accelerometer→nextAccelerometer() accelerometer.nextAccelerometer()

## YAccelerometer

Continues the enumeration of accelerometers started using `yFirstAccelerometer()`.

js	function <b>nextAccelerometer</b> ( )
nodejs	function <b>nextAccelerometer</b> ( )
php	function <b>nextAccelerometer</b> ( )
cpp	YAccelerometer * <b>nextAccelerometer</b> ( )
m	-(YAccelerometer*) <b>nextAccelerometer</b>
pas	function <b>nextAccelerometer</b> ( ): TYAccelerometer
vb	function <b>nextAccelerometer</b> ( ) As YAccelerometer
cs	YAccelerometer <b>nextAccelerometer</b> ( )
java	YAccelerometer <b>nextAccelerometer</b> ( )
py	def <b>nextAccelerometer</b> ( )

### Returns :

a pointer to a `YAccelerometer` object, corresponding to an accelerometer currently online, or a `null` pointer if there are no more accelerometers to enumerate.

## accelerometer→registerTimedReportCallback() accelerometer.registerTimedReportCallback()

YAccelerometer

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YAccelerometerTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YAccelerometerTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYAccelerometerTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## accelerometer→registerValueCallback() accelerometer.registerValueCallback()

## YAccelerometer

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YAccelerometerValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YAccelerometerValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYAccelerometerValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**accelerometer→set\_highestValue()**  
**accelerometer→setHighestValue()**  
**accelerometer.set\_highestValue()**

YAccelerometer

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YAccelerometer <b>target</b> <b>set_highestValue</b> <b>newval</b>

#### Parameters :

**newval** a floating point number corresponding to the recorded maximal value observed

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→set\_logFrequency()**  
**accelerometer→setLogFrequency()**  
**accelerometer.set\_logFrequency()**

**YAccelerometer**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YAccelerometer <b>target</b> <b>set_logFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the datalogger recording frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→set\_logicalName()**  
**accelerometer→setLogicalName()**  
**accelerometer.set\_logicalName()**

YAccelerometer

Changes the logical name of the accelerometer.

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

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 accelerometer.

#### Returns :

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.



**accelerometer→set\_lowestValue()**  
**accelerometer→setLowestValue()**  
**accelerometer.set\_lowestValue()**

**YAccelerometer**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YAccelerometer <b>target</b> <b>set_lowestValue</b> <b>newval</b>

#### Parameters :

**newval** a floating point number corresponding to the recorded minimal value observed

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→set\_reportFrequency()****YAccelerometer****accelerometer→setReportFrequency()****accelerometer.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YAccelerometer <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→set\_resolution()**  
**accelerometer→setResolution()**  
**accelerometer.set\_resolution()**

**YAccelerometer**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YAccelerometer <b>target</b> <b>set_resolution</b> <b>newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

#### Parameters :

**newval** a floating point number corresponding to the resolution of the measured physical values

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**accelerometer→set\_userdata()****YAccelerometer****accelerometer→setUserData()****accelerometer.set\_userdata()**

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

<code>js</code>	<code>function set_userdata( data)</code>
<code>nodejs</code>	<code>function set_userdata( data)</code>
<code>php</code>	<code>function set_userdata( \$data)</code>
<code>cpp</code>	<code>void set_userdata( void* data)</code>
<code>m</code>	<code>-(void) setUserData : (void*) data</code>
<code>pas</code>	<code>procedure set_userdata( data: Tobject)</code>
<code>vb</code>	<code>procedure set_userdata( ByVal data As Object)</code>
<code>cs</code>	<code>void set_userdata( object data)</code>
<code>java</code>	<code>void set_userdata( Object data)</code>
<code>py</code>	<code>def set_userdata( data)</code>

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

**accelerometer→wait\_async()****YAccelerometer**

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.

### 3.3. AnButton function interface

Yoctopuce application programming interface allows you to measure the state of a simple button as well as to read an analog potentiometer (variable resistance). This can be use for instance with a continuous rotating knob, a throttle grip or a joystick. The module is capable to calibrate itself on min and max values, in order to compute a calibrated value that varies proportionally with the potentiometer position, regardless of its total resistance.

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

js	<code>&lt;script type='text/javascript' src='yocto_anbutton.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YAnButton = yoctolib.YAnButton;</code>
php	<code>require_once('yocto_anbutton.php');</code>
c++	<code>#include "yocto_anbutton.h"</code>
m	<code>#import "yocto_anbutton.h"</code>
pas	<code>uses yocto_anbutton;</code>
vb	<code>yocto_anbutton.vb</code>
cs	<code>yocto_anbutton.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YAnButton;</code>
py	<code>from yocto_anbutton import *</code>

#### Global functions

##### **yFindAnButton(func)**

Retrieves an analog input for a given identifier.

##### **yFirstAnButton()**

Starts the enumeration of analog inputs currently accessible.

#### YAnButton methods

##### **anbutton→describe()**

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

##### **anbutton→get\_advertisedValue()**

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

##### **anbutton→get\_analogCalibration()**

Tells if a calibration process is currently ongoing.

##### **anbutton→get\_calibratedValue()**

Returns the current calibrated input value (between 0 and 1000, included).

##### **anbutton→get\_calibrationMax()**

Returns the maximal value measured during the calibration (between 0 and 4095, included).

##### **anbutton→get\_calibrationMin()**

Returns the minimal value measured during the calibration (between 0 and 4095, included).

##### **anbutton→get\_errorMessage()**

Returns the error message of the latest error with the analog input.

##### **anbutton→get\_errorType()**

Returns the numerical error code of the latest error with the analog input.

##### **anbutton→get\_friendlyName()**

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

##### **anbutton→get\_functionDescriptor()**

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

**anbutton→get\_functionId()**

Returns the hardware identifier of the analog input, without reference to the module.

**anbutton→get\_hardwareId()**

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

**anbutton→get\_isPressed()**

Returns true if the input (considered as binary) is active (closed contact), and false otherwise.

**anbutton→get\_lastTimePressed()**

Returns the number of elapsed milliseconds between the module power on and the last time the input button was pressed (the input contact transitionned from open to closed).

**anbutton→get\_lastTimeReleased()**

Returns the number of elapsed milliseconds between the module power on and the last time the input button was released (the input contact transitionned from closed to open).

**anbutton→get\_logicalName()**

Returns the logical name of the analog input.

**anbutton→get\_module()**

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

**anbutton→get\_module\_async(callback, context)**

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

**anbutton→get\_pulseCounter()**

Returns the pulse counter value

**anbutton→get\_pulseTimer()**

Returns the timer of the pulses counter (ms)

**anbutton→get\_rawValue()**

Returns the current measured input value as-is (between 0 and 4095, included).

**anbutton→get\_sensitivity()**

Returns the sensibility for the input (between 1 and 1000) for triggering user callbacks.

**anbutton→get\_userData()**

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

**anbutton→isOnline()**

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

**anbutton→isOnline\_async(callback, context)**

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

**anbutton→load(msValidity)**

Preloads the analog input cache with a specified validity duration.

**anbutton→load\_async(msValidity, callback, context)**

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

**anbutton→nextAnButton()**

Continues the enumeration of analog inputs started using `yFirstAnButton()`.

**anbutton→registerValueCallback(callback)**

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

**anbutton→resetCounter()**

Returns the pulse counter value as well as his timer

**anbutton→set\_analogCalibration(newval)**

Starts or stops the calibration process.

**anbutton→set\_calibrationMax(newval)**

### 3. Reference

Changes the maximal calibration value for the input (between 0 and 4095, included), without actually starting the automated calibration.

**anbutton**→**set\_calibrationMin**(newval)

Changes the minimal calibration value for the input (between 0 and 4095, included), without actually starting the automated calibration.

**anbutton**→**set\_logicalName**(newval)

Changes the logical name of the analog input.

**anbutton**→**set\_sensitivity**(newval)

Changes the sensibility for the input (between 1 and 1000) for triggering user callbacks.

**anbutton**→**set\_userData**(data)

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

**anbutton**→**wait\_async**(callback, context)

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



## YAnButton.FindAnButton() yFindAnButton()YAnButton.FindAnButton()

## YAnButton

Retrieves an analog input for a given identifier.

js	function <b>yFindAnButton</b> ( <b>func</b> )
nodejs	function <b>FindAnButton</b> ( <b>func</b> )
php	function <b>yFindAnButton</b> ( <b>\$func</b> )
cpp	YAnButton* <b>yFindAnButton</b> ( const string& <b>func</b> )
m	YAnButton* <b>yFindAnButton</b> ( NSString* <b>func</b> )
pas	function <b>yFindAnButton</b> ( <b>func</b> : string): TYAnButton
vb	function <b>yFindAnButton</b> ( ByVal <b>func</b> As String) As YAnButton
cs	YAnButton <b>FindAnButton</b> ( string <b>func</b> )
java	YAnButton <b>FindAnButton</b> ( String <b>func</b> )
py	def <b>FindAnButton</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the analog input is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YAnButton.isOnline()` to test if the analog input is indeed online at a given time. In case of ambiguity when looking for an analog input 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 analog input

### Returns :

a `YAnButton` object allowing you to drive the analog input.

**YAnButton.FirstAnButton()****YAnButton****yFirstAnButton()YAnButton.FirstAnButton()**

Starts the enumeration of analog inputs currently accessible.

js	function <b>yFirstAnButton</b> ( )
nodejs	function <b>FirstAnButton</b> ( )
php	function <b>yFirstAnButton</b> ( )
cpp	YAnButton* <b>yFirstAnButton</b> ( )
m	YAnButton* <b>yFirstAnButton</b> ( )
pas	function <b>yFirstAnButton</b> ( ): TYAnButton
vb	function <b>yFirstAnButton</b> ( ) As YAnButton
cs	YAnButton <b>FirstAnButton</b> ( )
java	YAnButton <b>FirstAnButton</b> ( )
py	def <b>FirstAnButton</b> ( )

Use the method `YAnButton.nextAnButton( )` to iterate on next analog inputs.

**Returns :**

a pointer to a `YAnButton` object, corresponding to the first analog input currently online, or a `null` pointer if there are none.

**anbutton→describe()****anbutton.describe()****YAnButton**

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

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

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(BadCustomName.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 analog input (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**anbutton→get\_advertisedValue()****YAnButton****anbutton→advertisedValue()****anbutton.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the analog input (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**anbutton→get\_analogCalibration()****YAnButton****anbutton→analogCalibration()****anbutton.get\_analogCalibration()**

Tells if a calibration process is currently ongoing.

js	function <b>get_analogCalibration</b> ( )
nodejs	function <b>get_analogCalibration</b> ( )
php	function <b>get_analogCalibration</b> ( )
cpp	Y_ANALOGCALIBRATION_enum <b>get_analogCalibration</b> ( )
m	-(Y_ANALOGCALIBRATION_enum) analogCalibration
pas	function <b>get_analogCalibration</b> ( ): Integer
vb	function <b>get_analogCalibration</b> ( ) As Integer
cs	int <b>get_analogCalibration</b> ( )
java	int <b>get_analogCalibration</b> ( )
py	def <b>get_analogCalibration</b> ( )
cmd	YAnButton <b>target</b> <b>get_analogCalibration</b>

**Returns :**

either Y\_ANALOGCALIBRATION\_OFF or Y\_ANALOGCALIBRATION\_ON

On failure, throws an exception or returns Y\_ANALOGCALIBRATION\_INVALID.

**anbutton**→**get\_calibratedValue()****YAnButton****anbutton**→**calibratedValue()****anbutton.get\_calibratedValue()**

Returns the current calibrated input value (between 0 and 1000, included).

js	function <b>get_calibratedValue</b> ( )
nodejs	function <b>get_calibratedValue</b> ( )
php	function <b>get_calibratedValue</b> ( )
cpp	int <b>get_calibratedValue</b> ( )
m	-(int) calibratedValue
pas	function <b>get_calibratedValue</b> ( ): LongInt
vb	function <b>get_calibratedValue</b> ( ) As Integer
cs	int <b>get_calibratedValue</b> ( )
java	int <b>get_calibratedValue</b> ( )
py	def <b>get_calibratedValue</b> ( )
cmd	YAnButton <b>target</b> <b>get_calibratedValue</b>

**Returns :**

an integer corresponding to the current calibrated input value (between 0 and 1000, included)

On failure, throws an exception or returns Y\_CALIBRATEDVALUE\_INVALID.

**anbutton→get\_calibrationMax()****YAnButton****anbutton→calibrationMax()****anbutton.get\_calibrationMax()**

Returns the maximal value measured during the calibration (between 0 and 4095, included).

js	function <b>get_calibrationMax</b> ( )
nodejs	function <b>get_calibrationMax</b> ( )
php	function <b>get_calibrationMax</b> ( )
cpp	int <b>get_calibrationMax</b> ( )
m	-(int) calibrationMax
pas	function <b>get_calibrationMax</b> ( ): LongInt
vb	function <b>get_calibrationMax</b> ( ) As Integer
cs	int <b>get_calibrationMax</b> ( )
java	int <b>get_calibrationMax</b> ( )
py	def <b>get_calibrationMax</b> ( )
cmd	YAnButton <b>target</b> <b>get_calibrationMax</b>

**Returns :**

an integer corresponding to the maximal value measured during the calibration (between 0 and 4095, included)

On failure, throws an exception or returns Y\_CALIBRATIONMAX\_INVALID.

**anbutton**→**get\_calibrationMin()****YAnButton****anbutton**→**calibrationMin()****anbutton.get\_calibrationMin()**

Returns the minimal value measured during the calibration (between 0 and 4095, included).

js	function <b>get_calibrationMin</b> ( )
nodejs	function <b>get_calibrationMin</b> ( )
php	function <b>get_calibrationMin</b> ( )
cpp	int <b>get_calibrationMin</b> ( )
m	-(int) calibrationMin
pas	function <b>get_calibrationMin</b> ( ): LongInt
vb	function <b>get_calibrationMin</b> ( ) As Integer
cs	int <b>get_calibrationMin</b> ( )
java	int <b>get_calibrationMin</b> ( )
py	def <b>get_calibrationMin</b> ( )
cmd	YAnButton <b>target</b> <b>get_calibrationMin</b>

**Returns :**

an integer corresponding to the minimal value measured during the calibration (between 0 and 4095, included)

On failure, throws an exception or returns Y\_CALIBRATIONMIN\_INVALID.



**anbutton→get\_errorMessage()****YAnButton****anbutton→errorMessage()****anbutton.get\_errorMessage()**

Returns the error message of the latest error with the analog input.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the analog input object

**anbutton**→**get\_errorType()****YAnButton****anbutton**→**errorType()****anbutton.get\_errorType()**

Returns the numerical error code of the latest error with the analog input.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the analog input object

**anbutton→get\_friendlyName()****YAnButton****anbutton→friendlyName()****anbutton.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the analog input using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**anbutton**→**get\_functionDescriptor()****YAnButton****anbutton**→**functionDescriptor()****anbutton.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**anbutton**→**get\_functionId()****YAnButton****anbutton**→**functionId()****anbutton.get\_functionId()**

Returns the hardware identifier of the analog input, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the analog input (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**anbutton**→**get\_hardwareId()****YAnButton****anbutton**→**hardwareId()****anbutton.get\_hardwareId()**

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

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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

**Returns :**

a string that uniquely identifies the analog input (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**anbutton→get\_isPressed()****YAnButton****anbutton→isPressed()****anbutton.get\_isPressed()**

Returns true if the input (considered as binary) is active (closed contact), and false otherwise.

js	function <b>get_isPressed</b> ( )
nodejs	function <b>get_isPressed</b> ( )
php	function <b>get_isPressed</b> ( )
cpp	Y_ISPRESSED_enum <b>get_isPressed</b> ( )
m	-(Y_ISPRESSED_enum) isPressed
pas	function <b>get_isPressed</b> ( ): Integer
vb	function <b>get_isPressed</b> ( ) As Integer
cs	int <b>get_isPressed</b> ( )
java	int <b>get_isPressed</b> ( )
py	def <b>get_isPressed</b> ( )
cmd	YAnButton <b>target</b> <b>get_isPressed</b>

**Returns :**

either Y\_ISPRESSED\_FALSE or Y\_ISPRESSED\_TRUE, according to true if the input (considered as binary) is active (closed contact), and false otherwise

On failure, throws an exception or returns Y\_ISPRESSED\_INVALID.

**anbutton→get\_lastTimePressed()****YAnButton****anbutton→lastTimePressed()****anbutton.get\_lastTimePressed()**

Returns the number of elapsed milliseconds between the module power on and the last time the input button was pressed (the input contact transitionned from open to closed).

js	function <b>get_lastTimePressed</b> ( )
nodejs	function <b>get_lastTimePressed</b> ( )
php	function <b>get_lastTimePressed</b> ( )
cpp	s64 <b>get_lastTimePressed</b> ( )
m	-(s64) lastTimePressed
pas	function <b>get_lastTimePressed</b> ( ): int64
vb	function <b>get_lastTimePressed</b> ( ) As Long
cs	long <b>get_lastTimePressed</b> ( )
java	long <b>get_lastTimePressed</b> ( )
py	def <b>get_lastTimePressed</b> ( )
cmd	YAnButton <b>target</b> <b>get_lastTimePressed</b>

**Returns :**

an integer corresponding to the number of elapsed milliseconds between the module power on and the last time the input button was pressed (the input contact transitionned from open to closed)

On failure, throws an exception or returns Y\_LASTTIMEPRESSED\_INVALID.



**anbutton→get\_lastTimeReleased()****YAnButton****anbutton→lastTimeReleased()****anbutton.get\_lastTimeReleased()**

Returns the number of elapsed milliseconds between the module power on and the last time the input button was released (the input contact transitionned from closed to open).

js	function <b>get_lastTimeReleased</b> ( )
nodejs	function <b>get_lastTimeReleased</b> ( )
php	function <b>get_lastTimeReleased</b> ( )
cpp	s64 <b>get_lastTimeReleased</b> ( )
m	-(s64) lastTimeReleased
pas	function <b>get_lastTimeReleased</b> ( ): int64
vb	function <b>get_lastTimeReleased</b> ( ) As Long
cs	long <b>get_lastTimeReleased</b> ( )
java	long <b>get_lastTimeReleased</b> ( )
py	def <b>get_lastTimeReleased</b> ( )
cmd	YAnButton <b>target</b> <b>get_lastTimeReleased</b>

**Returns :**

an integer corresponding to the number of elapsed milliseconds between the module power on and the last time the input button was released (the input contact transitionned from closed to open)

On failure, throws an exception or returns Y\_LASTTIMERELASED\_INVALID.

**anbutton**→**get\_logicalName()****YAnButton****anbutton**→**logicalName()****anbutton.get\_logicalName()**

Returns the logical name of the analog input.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YAnButton <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the analog input. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**anbutton**→**get\_module()****YAnButton****anbutton**→**module()****anbutton.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	<code>YModule *</code> <b>get_module</b> ( )
m	<code>-(YModule*)</code> module
pas	function <b>get_module</b> ( ): <code>TYModule</code>
vb	function <b>get_module</b> ( ) As <code>YModule</code>
cs	<code>YModule</code> <b>get_module</b> ( )
java	<code>YModule</code> <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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`

**anbutton**→**get\_module\_async()****YAnButton****anbutton**→**module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

---

**anbutton→get\_pulseCounter()****YAnButton****anbutton→pulseCounter()****anbutton.get\_pulseCounter()**

---

Returns the pulse counter value

js	function <b>get_pulseCounter</b> ( )
nodejs	function <b>get_pulseCounter</b> ( )
php	function <b>get_pulseCounter</b> ( )
cpp	s64 <b>get_pulseCounter</b> ( )
m	-(s64) pulseCounter
pas	function <b>get_pulseCounter</b> ( ): int64
vb	function <b>get_pulseCounter</b> ( ) As Long
cs	long <b>get_pulseCounter</b> ( )
java	long <b>get_pulseCounter</b> ( )
py	def <b>get_pulseCounter</b> ( )

**Returns :**

an integer corresponding to the pulse counter value

On failure, throws an exception or returns Y\_PULSECOUNTER\_INVALID.

**anbutton**→**get\_pulseTimer()****YAnButton****anbutton**→**pulseTimer()****anbutton.get\_pulseTimer()**

Returns the timer of the pulses counter (ms)

js	function <b>get_pulseTimer</b> ( )
nodejs	function <b>get_pulseTimer</b> ( )
php	function <b>get_pulseTimer</b> ( )
cpp	s64 <b>get_pulseTimer</b> ( )
m	-(s64) pulseTimer
pas	function <b>get_pulseTimer</b> ( ): int64
vb	function <b>get_pulseTimer</b> ( ) As Long
cs	long <b>get_pulseTimer</b> ( )
java	long <b>get_pulseTimer</b> ( )
py	def <b>get_pulseTimer</b> ( )

**Returns :**

an integer corresponding to the timer of the pulses counter (ms)

On failure, throws an exception or returns Y\_PULSETIMER\_INVALID.

**anbutton**→**get\_rawValue()****YAnButton****anbutton**→**rawValue()****anbutton.getRawValue()**

Returns the current measured input value as-is (between 0 and 4095, included).

js	function <b>get_rawValue</b> ( )
nodejs	function <b>get_rawValue</b> ( )
php	function <b>get_rawValue</b> ( )
cpp	int <b>get_rawValue</b> ( )
m	-(int) rawValue
pas	function <b>get_rawValue</b> ( ): LongInt
vb	function <b>get_rawValue</b> ( ) As Integer
cs	int <b>get_rawValue</b> ( )
java	int <b>get_rawValue</b> ( )
py	def <b>get_rawValue</b> ( )
cmd	YAnButton <b>target</b> <b>get_rawValue</b>

**Returns :**

an integer corresponding to the current measured input value as-is (between 0 and 4095, included)

On failure, throws an exception or returns Y\_RAWVALUE\_INVALID.

**anbutton**→**get\_sensitivity()****YAnButton****anbutton**→**sensitivity()****anbutton.get\_sensitivity()**

Returns the sensibility for the input (between 1 and 1000) for triggering user callbacks.

js	function <b>get_sensitivity</b> ( )
nodejs	function <b>get_sensitivity</b> ( )
php	function <b>get_sensitivity</b> ( )
cpp	int <b>get_sensitivity</b> ( )
m	-(int) sensitivity
pas	function <b>get_sensitivity</b> ( ): LongInt
vb	function <b>get_sensitivity</b> ( ) As Integer
cs	int <b>get_sensitivity</b> ( )
java	int <b>get_sensitivity</b> ( )
py	def <b>get_sensitivity</b> ( )
cmd	YAnButton <b>target</b> <b>get_sensitivity</b>

**Returns :**

an integer corresponding to the sensibility for the input (between 1 and 1000) for triggering user callbacks

On failure, throws an exception or returns Y\_SENSITIVITY\_INVALID.



**anbutton**→**get\_userData()****YAnButton****anbutton**→**userData()****anbutton.get\_userData()**

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

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

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.

**anbutton→isOnline()anbutton.isOnline()****YAnButton**

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

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

If there is a cached value for the analog input 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 analog input.

**Returns :**

`true` if the analog input can be reached, and `false` otherwise

**anbutton→isOnline\_async()****YAnButton**

Checks if the analog input 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 analog input 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.

**anbutton**→**load()****anbutton.load()****YAnButton**

Preloads the analog input cache with a specified validity duration.

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

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 traffic 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.

**anbutton→load\_async()****YAnButton**

Preloads the analog input 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 traffic 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.

**anbutton→nextAnButton()anbutton.nextAnButton()****YAnButton**

Continues the enumeration of analog inputs started using `yFirstAnButton()`.

js	function <b>nextAnButton</b> ( )
nodejs	function <b>nextAnButton</b> ( )
php	function <b>nextAnButton</b> ( )
cpp	YAnButton * <b>nextAnButton</b> ( )
m	-(YAnButton*) <b>nextAnButton</b>
pas	function <b>nextAnButton</b> ( ): TYAnButton
vb	function <b>nextAnButton</b> ( ) As YAnButton
cs	YAnButton <b>nextAnButton</b> ( )
java	YAnButton <b>nextAnButton</b> ( )
py	def <b>nextAnButton</b> ( )

**Returns :**

a pointer to a `YAnButton` object, corresponding to an analog input currently online, or a `null` pointer if there are no more analog inputs to enumerate.

## anbutton→registerValueCallback() anbutton.registerValueCallback()

YAnButton

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YAnButtonValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YAnButtonValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYAnButtonValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**anbutton→resetCounter()****anbutton.resetCounter()****YAnButton**

Returns the pulse counter value as well as his timer

js	function <b>resetCounter</b> ( )
nodejs	function <b>resetCounter</b> ( )
php	function <b>resetCounter</b> ( )
cpp	int <b>resetCounter</b> ( )
m	-(int) <b>resetCounter</b>
pas	function <b>resetCounter</b> ( ): LongInt
vb	function <b>resetCounter</b> ( ) As Integer
cs	int <b>resetCounter</b> ( )
java	int <b>resetCounter</b> ( )
py	def <b>resetCounter</b> ( )

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**anbutton→set\_analogCalibration()**  
**anbutton→setAnalogCalibration()**  
**anbutton.set\_analogCalibration()**

**YAnButton**

Starts or stops the calibration process.

js	function <b>set_analogCalibration</b> ( <b>newval</b> )
nodejs	function <b>set_analogCalibration</b> ( <b>newval</b> )
php	function <b>set_analogCalibration</b> ( <b>\$newval</b> )
cpp	int <b>set_analogCalibration</b> ( Y_ANALOGCALIBRATION_enum <b>newval</b> )
m	-(int) setAnalogCalibration : (Y_ANALOGCALIBRATION_enum) <b>newval</b>
pas	function <b>set_analogCalibration</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_analogCalibration</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_analogCalibration</b> ( int <b>newval</b> )
java	int <b>set_analogCalibration</b> ( int <b>newval</b> )
py	def <b>set_analogCalibration</b> ( <b>newval</b> )
cmd	YAnButton <b>target</b> <b>set_analogCalibration</b> <b>newval</b>

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

**Parameters :**

**newval** either Y\_ANALOGCALIBRATION\_OFF or Y\_ANALOGCALIBRATION\_ON

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**anbutton→set\_calibrationMax()****YAnButton****anbutton→setCalibrationMax()****anbutton.set\_calibrationMax()**

Changes the maximal calibration value for the input (between 0 and 4095, included), without actually starting the automated calibration.

js	function <b>set_calibrationMax</b> ( <b>newval</b> )
nodejs	function <b>set_calibrationMax</b> ( <b>newval</b> )
php	function <b>set_calibrationMax</b> ( <b>\$newval</b> )
cpp	int <b>set_calibrationMax</b> ( int <b>newval</b> )
m	-(int) setCalibrationMax : (int) <b>newval</b>
pas	function <b>set_calibrationMax</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_calibrationMax</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_calibrationMax</b> ( int <b>newval</b> )
java	int <b>set_calibrationMax</b> ( int <b>newval</b> )
py	def <b>set_calibrationMax</b> ( <b>newval</b> )
cmd	YAnButton <b>target set_calibrationMax newval</b>

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

**Parameters :**

**newval** an integer corresponding to the maximal calibration value for the input (between 0 and 4095, included), without actually starting the automated calibration

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**anbutton→set\_calibrationMin()**  
**anbutton→setCalibrationMin()**  
**anbutton.set\_calibrationMin()**

**YAnButton**

Changes the minimal calibration value for the input (between 0 and 4095, included), without actually starting the automated calibration.

js	function <b>set_calibrationMin</b> ( <b>newval</b> )
nodejs	function <b>set_calibrationMin</b> ( <b>newval</b> )
php	function <b>set_calibrationMin</b> ( <b>\$newval</b> )
cpp	int <b>set_calibrationMin</b> ( int <b>newval</b> )
m	-(int) setCalibrationMin : (int) <b>newval</b>
pas	function <b>set_calibrationMin</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_calibrationMin</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_calibrationMin</b> ( int <b>newval</b> )
java	int <b>set_calibrationMin</b> ( int <b>newval</b> )
py	def <b>set_calibrationMin</b> ( <b>newval</b> )
cmd	YAnButton <b>target</b> <b>set_calibrationMin</b> <b>newval</b>

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

#### Parameters :

**newval** an integer corresponding to the minimal calibration value for the input (between 0 and 4095, included), without actually starting the automated calibration

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**anbutton→set\_logicalName()****YAnButton****anbutton→setLogicalName()****anbutton.set\_logicalName()**

Changes the logical name of the analog input.

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

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 analog input.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**anbutton→set\_sensitivity()****YAnButton****anbutton→setSensitivity()anbutton.set\_sensitivity()**

Changes the sensibility for the input (between 1 and 1000) for triggering user callbacks.

js	function <b>set_sensitivity</b> ( <b>newval</b> )
nodejs	function <b>set_sensitivity</b> ( <b>newval</b> )
php	function <b>set_sensitivity</b> ( <b>\$newval</b> )
cpp	int <b>set_sensitivity</b> ( int <b>newval</b> )
m	-(int) <b>setSensitivity</b> : (int) <b>newval</b>
pas	function <b>set_sensitivity</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_sensitivity</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_sensitivity</b> ( int <b>newval</b> )
java	int <b>set_sensitivity</b> ( int <b>newval</b> )
py	def <b>set_sensitivity</b> ( <b>newval</b> )
cmd	YAnButton <b>target set_sensitivity newval</b>

The sensibility is used to filter variations around a fixed value, but does not preclude the transmission of events when the input value evolves constantly in the same direction. Special case: when the value 1000 is used, the callback will only be thrown when the logical state of the input switches from pressed to released and back. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

**Parameters :**

**newval** an integer corresponding to the sensibility for the input (between 1 and 1000) for triggering user callbacks

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**anbutton**→**set\_userData()****YAnButton****anbutton**→**setUserData()****anbutton.set\_userData()**

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

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

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

**anbutton→wait\_async()****YAnButton**

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.

## 3.4. CarbonDioxide function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindCarbonDioxide(func)

Retrieves a CO2 sensor for a given identifier.

#### yFirstCarbonDioxide()

Starts the enumeration of CO2 sensors currently accessible.

### YCarbonDioxide methods

#### carbondioxide→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### carbondioxide→describe()

Returns a short text that describes unambiguously the instance of the CO2 sensor in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### carbondioxide→get\_advertisedValue()

Returns the current value of the CO2 sensor (no more than 6 characters).

#### carbondioxide→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### carbondioxide→get\_currentValue()

Returns the current value of the CO2 concentration.

#### carbondioxide→get\_errorMessage()

Returns the error message of the latest error with the CO2 sensor.

#### carbondioxide→get\_errorType()

Returns the numerical error code of the latest error with the CO2 sensor.

#### carbondioxide→get\_friendlyName()

Returns a global identifier of the CO2 sensor in the format MODULE\_NAME . FUNCTION\_NAME.

#### carbondioxide→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### carbondioxide→get\_functionId()

Returns the hardware identifier of the CO2 sensor, without reference to the module.

#### carbondioxide→get\_hardwareId()

Returns the unique hardware identifier of the CO2 sensor in the form SERIAL . FUNCTIONID.



**carbondioxide→get\_highestValue()**

Returns the maximal value observed for the CO2 concentration since the device was started.

**carbondioxide→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**carbondioxide→get\_logicalName()**

Returns the logical name of the CO2 sensor.

**carbondioxide→get\_lowestValue()**

Returns the minimal value observed for the CO2 concentration since the device was started.

**carbondioxide→get\_module()**

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

**carbondioxide→get\_module\_async(callback, context)**

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

**carbondioxide→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**carbondioxide→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**carbondioxide→get\_resolution()**

Returns the resolution of the measured values.

**carbondioxide→get\_unit()**

Returns the measuring unit for the CO2 concentration.

**carbondioxide→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**carbondioxide→isOnline()**

Checks if the CO2 sensor is currently reachable, without raising any error.

**carbondioxide→isOnline\_async(callback, context)**

Checks if the CO2 sensor is currently reachable, without raising any error (asynchronous version).

**carbondioxide→load(msValidity)**

Preloads the CO2 sensor cache with a specified validity duration.

**carbondioxide→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**carbondioxide→load\_async(msValidity, callback, context)**

Preloads the CO2 sensor cache with a specified validity duration (asynchronous version).

**carbondioxide→nextCarbonDioxide()**

Continues the enumeration of CO2 sensors started using yFirstCarbonDioxide( ).

**carbondioxide→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**carbondioxide→registerValueCallback(callback)**

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

**carbondioxide→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**carbondioxide→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**carbondioxide→set\_logicalName(newval)**

Changes the logical name of the CO2 sensor.

### 3. Reference

---

**carbondioxide**→**set\_lowestValue**(newval)

Changes the recorded minimal value observed.

**carbondioxide**→**set\_reportFrequency**(newval)

Changes the timed value notification frequency for this function.

**carbondioxide**→**set\_resolution**(newval)

Changes the resolution of the measured physical values.

**carbondioxide**→**set\_userData**(data)

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

**carbondioxide**→**wait\_async**(callback, context)

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

## YCarbonDioxide.FindCarbonDioxide() yFindCarbonDioxide() YCarbonDioxide.FindCarbonDioxide()

## YCarbonDioxide

Retrieves a CO2 sensor for a given identifier.

js	function <b>yFindCarbonDioxide</b> ( <b>func</b> )
nodejs	function <b>FindCarbonDioxide</b> ( <b>func</b> )
php	function <b>yFindCarbonDioxide</b> ( <b>\$func</b> )
cpp	YCarbonDioxide* <b>yFindCarbonDioxide</b> ( const string& <b>func</b> )
m	YCarbonDioxide* <b>yFindCarbonDioxide</b> ( NSString* <b>func</b> )
pas	function <b>yFindCarbonDioxide</b> ( <b>func</b> : string): TYCarbonDioxide
vb	function <b>yFindCarbonDioxide</b> ( ByVal <b>func</b> As String) As YCarbonDioxide
cs	YCarbonDioxide <b>FindCarbonDioxide</b> ( string <b>func</b> )
java	YCarbonDioxide <b>FindCarbonDioxide</b> ( String <b>func</b> )
py	def <b>FindCarbonDioxide</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the CO2 sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YCarbonDioxide.isOnline()` to test if the CO2 sensor is indeed online at a given time. In case of ambiguity when looking for a CO2 sensor 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 CO2 sensor

### Returns :

a YCarbonDioxide object allowing you to drive the CO2 sensor.

## **YCarbonDioxide.FirstCarbonDioxide() yFirstCarbonDioxide() YCarbonDioxide.FirstCarbonDioxide()**

## **YCarbonDioxide**

Starts the enumeration of CO2 sensors currently accessible.

<code>js</code>	<code>function <b>yFirstCarbonDioxide</b>( )</code>
<code>nodejs</code>	<code>function <b>FirstCarbonDioxide</b>( )</code>
<code>php</code>	<code>function <b>yFirstCarbonDioxide</b>( )</code>
<code>cpp</code>	<code>YCarbonDioxide* <b>yFirstCarbonDioxide</b>( )</code>
<code>m</code>	<code>YCarbonDioxide* <b>yFirstCarbonDioxide</b>( )</code>
<code>pas</code>	<code>function <b>yFirstCarbonDioxide</b>( ): TYCarbonDioxide</code>
<code>vb</code>	<code>function <b>yFirstCarbonDioxide</b>( ) As YCarbonDioxide</code>
<code>cs</code>	<code>YCarbonDioxide <b>FirstCarbonDioxide</b>( )</code>
<code>java</code>	<code>YCarbonDioxide <b>FirstCarbonDioxide</b>( )</code>
<code>py</code>	<code>def <b>FirstCarbonDioxide</b>( )</code>

Use the method `YCarbonDioxide.nextCarbonDioxide( )` to iterate on next CO2 sensors.

### **Returns :**

a pointer to a `YCarbonDioxide` object, corresponding to the first CO2 sensor currently online, or a null pointer if there are none.

## carbondioxide→calibrateFromPoints() carbondioxide.calibrateFromPoints()

YCarbonDioxide

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js      function calibrateFromPoints( rawValues, refValues)
node.js function calibrateFromPoints( rawValues, refValues)
php     function calibrateFromPoints( $rawValues, $refValues)
cpp     int calibrateFromPoints( vector<double> rawValues,
                                vector<double> refValues)

m       -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function calibrateFromPoints( rawValues: TDoubleArray,
                                refValues: TDoubleArray): LongInt

vb      procedure calibrateFromPoints( )

cs      int calibrateFromPoints( List<double> rawValues,
                                List<double> refValues)

java    int calibrateFromPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def calibrateFromPoints( rawValues, refValues)

cmd     YCarbonDioxide target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**carbondioxide→describe()**carbondioxide.describe()**YCarbonDioxide**

Returns a short text that describes unambiguously the instance of the CO2 sensor in the form  
`TYPE (NAME) =SERIAL.FUNCTIONID`.

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

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 CO2 sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**carbondioxide→get\_advertisedValue()****YCarbonDioxide****carbondioxide→advertisedValue()****carbondioxide.get\_advertisedValue()**

Returns the current value of the CO2 sensor (no more than 6 characters).

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

**Returns :**

a string corresponding to the current value of the CO2 sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**carbondioxide→get\_currentRawValue()**  
**carbondioxide→currentRawValue()**  
**carbondioxide.get\_currentRawValue()**

**YCarbonDioxide**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.



**carbondioxide→get\_currentValue()****YCarbonDioxide****carbondioxide→currentValue()****carbondioxide.get\_currentValue()**

Returns the current value of the CO2 concentration.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the CO2 concentration

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**carbondioxide→get\_errorMessage()**  
**carbondioxide→errorMessage()**  
**carbondioxide.get\_errorMessage()**

**YCarbonDioxide**

Returns the error message of the latest error with the CO2 sensor.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the CO2 sensor object

**carbondioxide→get\_errorType()**  
**carbondioxide→errorType()**  
**carbondioxide.get\_errorType()**

**YCarbonDioxide**

Returns the numerical error code of the latest error with the CO2 sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the CO2 sensor object

**carbondioxide→get\_friendlyName()**  
**carbondioxide→friendlyName()**  
**carbondioxide.get\_friendlyName()**

**YCarbonDioxide**

Returns a global identifier of the CO2 sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the CO2 sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the CO2 sensor (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the CO2 sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**carbondioxide→get\_functionDescriptor()**  
**carbondioxide→functionDescriptor()**  
**carbondioxide.get\_functionDescriptor()**

**YCarbonDioxide**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**carbondioxide→get\_functionId()**  
**carbondioxide→functionId()**  
**carbondioxide.get\_functionId()**

---

**YCarbonDioxide**

Returns the hardware identifier of the CO2 sensor, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the CO2 sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**carbondioxide→get\_hardwareId()**  
**carbondioxide→hardwareId()**  
**carbondioxide.get\_hardwareId()**

**YCarbonDioxide**

Returns the unique hardware identifier of the CO2 sensor in the form `SERIAL.FUNCTIONID`.

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the CO2 sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the CO2 sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**carbondioxide→get\_highestValue()**  
**carbondioxide→highestValue()**  
**carbondioxide.get\_highestValue()**

**YCarbonDioxide**

Returns the maximal value observed for the CO2 concentration since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the CO2 concentration since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.



carbondioxide→get\_logFrequency()  
 carbondioxide→logFrequency()  
 carbondioxide.get\_logFrequency()

YCarbonDioxide

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_logFrequency</b>

#### Returns :

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**carbondioxide→get\_logicalName()**  
**carbondioxide→logicalName()**  
**carbondioxide.get\_logicalName()**

**YCarbonDioxide**

Returns the logical name of the CO2 sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the CO2 sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**carbondioxide→get\_lowestValue()****YCarbonDioxide****carbondioxide→lowestValue()****carbondioxide.get\_lowestValue()**

Returns the minimal value observed for the CO2 concentration since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the CO2 concentration since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**carbondioxide→get\_module()**  
**carbondioxide→module()**  
**carbondioxide.get\_module()**

**YCarbonDioxide**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**carbondioxide→get\_module\_async()**  
**carbondioxide→module\_async()**

**YCarbonDioxide**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**carbondioxide→get\_recordedData()**  
**carbondioxide→recordedData()**  
**carbondioxide.get\_recordedData()**

YCarbonDioxide

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
c++	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YCarbonDioxide <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

#### Parameters :

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

#### Returns :

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**carbondioxide→get\_reportFrequency()****YCarbonDioxide****carbondioxide→reportFrequency()****carbondioxide.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**carbondioxide→get\_resolution()**  
**carbondioxide→resolution()**  
**carbondioxide.get\_resolution()**

**YCarbonDioxide**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.



**carbondioxide**→**get\_unit()****YCarbonDioxide****carbondioxide**→**unit()****carbondioxide.get\_unit()**

Returns the measuring unit for the CO2 concentration.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YCarbonDioxide <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the CO2 concentration

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**carbondioxide→get\_userdata()**  
**carbondioxide→userData()**  
**carbondioxide.get\_userdata()**

**YCarbonDioxide**

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

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**carbondioxide→isOnline()carbondioxide.isOnline()****YCarbonDioxide**

Checks if the CO2 sensor is currently reachable, without raising any error.

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

If there is a cached value for the CO2 sensor 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 CO2 sensor.

**Returns :**

`true` if the CO2 sensor can be reached, and `false` otherwise

**carbondioxide→isOnline\_async()****YCarbonDioxide**

Checks if the CO2 sensor 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 CO2 sensor 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.

**carbondioxide→load()**carbondioxide.load()**YCarbonDioxide**

Preloads the CO2 sensor cache with a specified validity duration.

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

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 traffic 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.

## carbondioxide→loadCalibrationPoints() carbondioxide.loadCalibrationPoints()

YCarbonDioxide

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                    var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YCarbonDioxide target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**carbondioxide→load\_async()****YCarbonDioxide**

Preloads the CO2 sensor 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 traffic 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.

**carbondioxide→nextCarbonDioxide()  
carbondioxide.nextCarbonDioxide()****YCarbonDioxide**

Continues the enumeration of CO2 sensors started using `yFirstCarbonDioxide()`.

js	function <b>nextCarbonDioxide</b> ( )
nodejs	function <b>nextCarbonDioxide</b> ( )
php	function <b>nextCarbonDioxide</b> ( )
cpp	YCarbonDioxide * <b>nextCarbonDioxide</b> ( )
m	-(YCarbonDioxide*) <b>nextCarbonDioxide</b>
pas	function <b>nextCarbonDioxide</b> ( ): TYCarbonDioxide
vb	function <b>nextCarbonDioxide</b> ( ) As YCarbonDioxide
cs	YCarbonDioxide <b>nextCarbonDioxide</b> ( )
java	YCarbonDioxide <b>nextCarbonDioxide</b> ( )
py	def <b>nextCarbonDioxide</b> ( )

**Returns :**

a pointer to a YCarbonDioxide object, corresponding to a CO2 sensor currently online, or a `null` pointer if there are no more CO2 sensors to enumerate.



## carbondioxide→registerTimedReportCallback() carbondioxide.registerTimedReportCallback()

## YCarbonDioxide

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YCarbonDioxideTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YCarbonDioxideTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYCarbonDioxideTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## carbondioxide→registerValueCallback() carbondioxide.registerValueCallback()

YCarbonDioxide

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YCarbonDioxideValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YCarbonDioxideValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYCarbonDioxideValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

carbondioxide→set\_highestValue()  
 carbondioxide→setHighestValue()  
 carbondioxide.set\_highestValue()

YCarbonDioxide

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YCarbonDioxide <b>target</b> <b>set_highestValue</b> <b>newval</b>

#### Parameters :

**newval** a floating point number corresponding to the recorded maximal value observed

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**carbondioxide→set\_logFrequency()**  
**carbondioxide→setLogFrequency()**  
**carbondioxide.set\_logFrequency()**

YCarbonDioxide

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YCarbonDioxide <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the datalogger recording frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**carbondioxide→set\_logicalName()****YCarbonDioxide****carbondioxide→setLogicalName()****carbondioxide.set\_logicalName()**

Changes the logical name of the CO2 sensor.

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

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 CO2 sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**carbondioxide→set\_lowestValue()**  
**carbondioxide→setLowestValue()**  
**carbondioxide.set\_lowestValue()**

**YCarbonDioxide**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YCarbonDioxide <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

carbondioxide→set\_reportFrequency()  
carbondioxide→setReportFrequency()  
carbondioxide.set\_reportFrequency()

YCarbonDioxide

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YCarbonDioxide <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**carbondioxide→set\_resolution()**  
**carbondioxide→setResolution()**  
**carbondioxide.set\_resolution()**

**YCarbonDioxide**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YCarbonDioxide <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**carbondioxide→set\_userData()**  
**carbondioxide→setUserData()**  
**carbondioxide.set\_userData()**

**YCarbonDioxide**

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

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

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

**carbondioxide→wait\_async()****YCarbonDioxide**

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.

## 3.5. ColorLed function interface

Yoctopuce application programming interface allows you to drive a color led using RGB coordinates as well as HSL coordinates. The module performs all conversions from RGB to HSL automatically. It is then self-evident to turn on a led with a given hue and to progressively vary its saturation or lightness. If needed, you can find more information on the difference between RGB and HSL in the section following this one.

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

js	<code>&lt;script type='text/javascript' src='yocto_colorled.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib'); var YColorLed = yoctolib.YColorLed;</code>
php	<code>require_once('yocto_colorled.php');</code>
c++	<code>#include "yocto_colorled.h"</code>
m	<code>#import "yocto_colorled.h"</code>
pas	<code>uses yocto_colorled;</code>
vb	<code>yocto_colorled.vb</code>
cs	<code>yocto_colorled.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YColorLed;</code>
py	<code>from yocto_colorled import *</code>

### Global functions

#### **yFindColorLed(func)**

Retrieves an RGB led for a given identifier.

#### **yFirstColorLed()**

Starts the enumeration of RGB leds currently accessible.

### YColorLed methods

#### **colorled→describe()**

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

#### **colorled→get\_advertisedValue()**

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

#### **colorled→get\_errorMessage()**

Returns the error message of the latest error with the RGB led.

#### **colorled→get\_errorType()**

Returns the numerical error code of the latest error with the RGB led.

#### **colorled→get\_friendlyName()**

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

#### **colorled→get\_functionDescriptor()**

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

#### **colorled→get\_functionId()**

Returns the hardware identifier of the RGB led, without reference to the module.

#### **colorled→get\_hardwareId()**

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

#### **colorled→get\_hslColor()**

Returns the current HSL color of the led.

#### **colorled→get\_logicalName()**

Returns the logical name of the RGB led.

**colorled→get\_module()**

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

**colorled→get\_module\_async(callback, context)**

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

**colorled→get\_rgbColor()**

Returns the current RGB color of the led.

**colorled→get\_rgbColorAtPowerOn()**

Returns the configured color to be displayed when the module is turned on.

**colorled→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**colorled→hslMove(hsl\_target, ms\_duration)**

Performs a smooth transition in the HSL color space between the current color and a target color.

**colorled→isOnline()**

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

**colorled→isOnline\_async(callback, context)**

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

**colorled→load(msValidity)**

Preloads the RGB led cache with a specified validity duration.

**colorled→load\_async(msValidity, callback, context)**

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

**colorled→nextColorLed()**

Continues the enumeration of RGB leds started using yFirstColorLed( ).

**colorled→registerValueCallback(callback)**

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

**colorled→rgbMove(rgb\_target, ms\_duration)**

Performs a smooth transition in the RGB color space between the current color and a target color.

**colorled→set\_hslColor(newval)**

Changes the current color of the led, using a color HSL.

**colorled→set\_logicalName(newval)**

Changes the logical name of the RGB led.

**colorled→set\_rgbColor(newval)**

Changes the current color of the led, using a RGB color.

**colorled→set\_rgbColorAtPowerOn(newval)**

Changes the color that the led will display by default when the module is turned on.

**colorled→set\_userData(data)**

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

**colorled→wait\_async(callback, context)**

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

## YColorLed.FindColorLed() yFindColorLed()YColorLed.FindColorLed()

## YColorLed

Retrieves an RGB led for a given identifier.

js	function <b>yFindColorLed</b> ( <b>func</b> )
nodejs	function <b>FindColorLed</b> ( <b>func</b> )
php	function <b>yFindColorLed</b> ( <b>\$func</b> )
cpp	YColorLed* <b>yFindColorLed</b> ( const string& <b>func</b> )
m	YColorLed* <b>yFindColorLed</b> ( NSString* <b>func</b> )
pas	function <b>yFindColorLed</b> ( <b>func</b> : string): TYColorLed
vb	function <b>yFindColorLed</b> ( ByVal <b>func</b> As String) As YColorLed
cs	YColorLed <b>FindColorLed</b> ( string <b>func</b> )
java	YColorLed <b>FindColorLed</b> ( String <b>func</b> )
py	def <b>FindColorLed</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the RGB led is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YColorLed.isOnline()` to test if the RGB led is indeed online at a given time. In case of ambiguity when looking for an RGB led 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 RGB led

### Returns :

a `YColorLed` object allowing you to drive the RGB led.

**YColorLed.FirstColorLed()****YColorLed****yFirstColorLed()YColorLed.FirstColorLed()**

Starts the enumeration of RGB leds currently accessible.

js	function <b>yFirstColorLed</b> ( )
nodejs	function <b>FirstColorLed</b> ( )
php	function <b>yFirstColorLed</b> ( )
cpp	YColorLed* <b>yFirstColorLed</b> ( )
m	YColorLed* <b>yFirstColorLed</b> ( )
pas	function <b>yFirstColorLed</b> ( ): TYColorLed
vb	function <b>yFirstColorLed</b> ( ) As YColorLed
cs	YColorLed <b>FirstColorLed</b> ( )
java	YColorLed <b>FirstColorLed</b> ( )
py	def <b>FirstColorLed</b> ( )

Use the method `YColorLed.nextColorLed( )` to iterate on next RGB leds.

**Returns :**

a pointer to a `YColorLed` object, corresponding to the first RGB led currently online, or a `null` pointer if there are none.

**colorled→describe()colorled.describe()****YColorLed**

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

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

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(BadCustomName.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 RGB led (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**colored**→**get\_advertisedValue()****YColorLed****colored**→**advertisedValue()****colored.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the RGB led (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.



---

**colorled**→**get\_errorMessage()****YColorLed****colorled**→**errorMessage()****colorled.get\_errorMessage()**

---

Returns the error message of the latest error with the RGB led.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the RGB led object

**colorled**→**get\_errorType()****YColorLed****colorled**→**errorType()****colorled.get\_errorType()**

Returns the numerical error code of the latest error with the RGB led.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the RGB led object

**colorled**→**get\_friendlyName()****YColorLed****colorled**→**friendlyName()****colorled.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the RGB led using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**colored**→**get\_functionDescriptor()****colored**→**functionDescriptor()****colored.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**colorled**→**get\_functionId()****YColorLed****colorled**→**functionId()****colorled.get\_functionId()**

Returns the hardware identifier of the RGB led, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the RGB led (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**YColorLed**  
**YColorLed→get\_hardwareId()****YColorLed****YColorLed→hardwareId()**  
**YColorLed.get\_hardwareId()**

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

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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

**Returns :**

a string that uniquely identifies the RGB led (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**colorled**→**get\_hslColor()****YColorLed****colorled**→**hslColor()****colorled.get\_hslColor()**

Returns the current HSL color of the led.

js	function <b>get_hslColor</b> ( )
nodejs	function <b>get_hslColor</b> ( )
php	function <b>get_hslColor</b> ( )
cpp	int <b>get_hslColor</b> ( )
m	-(int) hslColor
pas	function <b>get_hslColor</b> ( ): LongInt
vb	function <b>get_hslColor</b> ( ) As Integer
cs	int <b>get_hslColor</b> ( )
java	int <b>get_hslColor</b> ( )
py	def <b>get_hslColor</b> ( )
cmd	YColorLed <b>target</b> <b>get_hslColor</b>

**Returns :**

an integer corresponding to the current HSL color of the led

On failure, throws an exception or returns Y\_HSLCOLOR\_INVALID.

**colored**→**get\_logicalName()****YColorLed****colored**→**logicalName()****colored.get\_logicalName()**

Returns the logical name of the RGB led.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YColorLed <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the RGB led. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**colorled**→**get\_module()****YColorLed****colorled**→**module()****colorled.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**colored**→**get\_module\_async()**  
**colored**→**module\_async()**

**YColorLed**

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.

**colorled**→**get\_rgbColor()****YColorLed****colorled**→**rgbColor()****colorled.get\_rgbColor()**

Returns the current RGB color of the led.

js	function <b>get_rgbColor</b> ( )
nodejs	function <b>get_rgbColor</b> ( )
php	function <b>get_rgbColor</b> ( )
cpp	int <b>get_rgbColor</b> ( )
m	-(int) rgbColor
pas	function <b>get_rgbColor</b> ( ): LongInt
vb	function <b>get_rgbColor</b> ( ) As Integer
cs	int <b>get_rgbColor</b> ( )
java	int <b>get_rgbColor</b> ( )
py	def <b>get_rgbColor</b> ( )
cmd	YColorLed <b>target</b> <b>get_rgbColor</b>

**Returns :**

an integer corresponding to the current RGB color of the led

On failure, throws an exception or returns Y\_RGBCOLOR\_INVALID.

**colored**→**get\_rgbColorAtPowerOn()**  
**colored**→**rgbColorAtPowerOn()**  
**colored.get\_rgbColorAtPowerOn()**

**YColorLed**

Returns the configured color to be displayed when the module is turned on.

js	function <b>get_rgbColorAtPowerOn</b> ( )
nodejs	function <b>get_rgbColorAtPowerOn</b> ( )
php	function <b>get_rgbColorAtPowerOn</b> ( )
cpp	int <b>get_rgbColorAtPowerOn</b> ( )
m	-(int) rgbColorAtPowerOn
pas	function <b>get_rgbColorAtPowerOn</b> ( ): LongInt
vb	function <b>get_rgbColorAtPowerOn</b> ( ) As Integer
cs	int <b>get_rgbColorAtPowerOn</b> ( )
java	int <b>get_rgbColorAtPowerOn</b> ( )
py	def <b>get_rgbColorAtPowerOn</b> ( )
cmd	YColorLed <b>target</b> <b>get_rgbColorAtPowerOn</b>

**Returns :**

an integer corresponding to the configured color to be displayed when the module is turned on

On failure, throws an exception or returns Y\_RGBCOLORATPOWERON\_INVALID.

**colorled**→**get\_userData()****YColorLed****colorled**→**userData()****colorled.get\_userData()**

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

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

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.

**colored**→**hslMove()****colored.hslMove()****YColorLed**

Performs a smooth transition in the HSL color space between the current color and a target color.

js	function <b>hslMove</b> ( <b>hsl_target</b> , <b>ms_duration</b> )
nodejs	function <b>hslMove</b> ( <b>hsl_target</b> , <b>ms_duration</b> )
php	function <b>hslMove</b> ( <b>\$hsl_target</b> , <b>\$ms_duration</b> )
cpp	int <b>hslMove</b> ( int <b>hsl_target</b> , int <b>ms_duration</b> )
m	-(int) <b>hslMove</b> : (int) <b>hsl_target</b> : (int) <b>ms_duration</b>
pas	function <b>hslMove</b> ( <b>hsl_target</b> : LongInt, <b>ms_duration</b> : LongInt): integer
vb	function <b>hslMove</b> ( ByVal <b>hsl_target</b> As Integer, ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>hslMove</b> ( int <b>hsl_target</b> , int <b>ms_duration</b> )
java	int <b>hslMove</b> ( int <b>hsl_target</b> , int <b>ms_duration</b> )
py	def <b>hslMove</b> ( <b>hsl_target</b> , <b>ms_duration</b> )
cmd	YColorLed <b>target</b> <b>hslMove</b> <b>hsl_target</b> <b>ms_duration</b>

**Parameters :**

**hsl\_target**    desired HSL color at the end of the transition  
**ms\_duration**   duration of the transition, in millisecond

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**colorled→isOnline()colorled.isOnline()****YColorLed**

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

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

If there is a cached value for the RGB led 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 RGB led.

**Returns :**

`true` if the RGB led can be reached, and `false` otherwise

**colorled→isOnline\_async()****YColorLed**

Checks if the RGB led 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 RGB led 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.



**colored→load()colored.load()****YColorLed**

Preloads the RGB led cache with a specified validity duration.

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

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 traffic 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.

**colorled→load\_async()****YColorLed**

Preloads the RGB led 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 traffic 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.

**colorled→nextColorLed()colorled.nextColorLed()****YColorLed**

Continues the enumeration of RGB leds started using `yFirstColorLed()`.

js	function <b>nextColorLed</b> ( )
nodejs	function <b>nextColorLed</b> ( )
php	function <b>nextColorLed</b> ( )
cpp	YColorLed * <b>nextColorLed</b> ( )
m	-(YColorLed*) <b>nextColorLed</b>
pas	function <b>nextColorLed</b> ( ): TYColorLed
vb	function <b>nextColorLed</b> ( ) As YColorLed
cs	YColorLed <b>nextColorLed</b> ( )
java	YColorLed <b>nextColorLed</b> ( )
py	def <b>nextColorLed</b> ( )

**Returns :**

a pointer to a `YColorLed` object, corresponding to an RGB led currently online, or a `null` pointer if there are no more RGB leds to enumerate.

## colored→registerValueCallback()

### colored.registerValueCallback()

YColorLed

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YColorLedValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YColorLedValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYColorLedValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**colorled→rgbMove()colorled.rgbMove()****YColorLed**

Performs a smooth transition in the RGB color space between the current color and a target color.

<code>js</code>	<code>function rgbMove( rgb_target, ms_duration)</code>
<code>nodejs</code>	<code>function rgbMove( rgb_target, ms_duration)</code>
<code>php</code>	<code>function rgbMove( \$rgb_target, \$ms_duration)</code>
<code>cpp</code>	<code>int rgbMove( int rgb_target, int ms_duration)</code>
<code>m</code>	<code>-(int) rgbMove : (int) rgb_target : (int) ms_duration</code>
<code>pas</code>	<code>function rgbMove( rgb_target: LongInt, ms_duration: LongInt): integer</code>
<code>vb</code>	<code>function rgbMove( ByVal rgb_target As Integer, ByVal ms_duration As Integer) As Integer</code>
<code>cs</code>	<code>int rgbMove( int rgb_target, int ms_duration)</code>
<code>java</code>	<code>int rgbMove( int rgb_target, int ms_duration)</code>
<code>py</code>	<code>def rgbMove( rgb_target, ms_duration)</code>
<code>cmd</code>	<code>YColorLed target rgbMove rgb_target ms_duration</code>

**Parameters :**

**rgb\_target** desired RGB color at the end of the transition

**ms\_duration** duration of the transition, in millisecond

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**colored**→**set\_hslColor()****YColorLed****colored**→**setHslColor()****colored.set\_hslColor()**

Changes the current color of the led, using a color HSL.

js	function <b>set_hslColor</b> ( <b>newval</b> )
nodejs	function <b>set_hslColor</b> ( <b>newval</b> )
php	function <b>set_hslColor</b> ( <b>\$newval</b> )
cpp	int <b>set_hslColor</b> ( int <b>newval</b> )
m	-(int) setHslColor : (int) <b>newval</b>
pas	function <b>set_hslColor</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_hslColor</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_hslColor</b> ( int <b>newval</b> )
java	int <b>set_hslColor</b> ( int <b>newval</b> )
py	def <b>set_hslColor</b> ( <b>newval</b> )
cmd	YColorLed <b>target set_hslColor newval</b>

Encoding is done as follows: 0xHHSSL.

**Parameters :**

**newval** an integer corresponding to the current color of the led, using a color HSL

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**colorled**→**set\_logicalName()****YColorLed****colorled**→**setLogicalName()****colorled.set\_logicalName()**

Changes the logical name of the RGB led.

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

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 RGB led.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**colored**→**set\_rgbColor()****YColorLed****colored**→**setRgbColor()****colored.set\_rgbColor()**

Changes the current color of the led, using a RGB color.

js	function <b>set_rgbColor</b> ( <b>newval</b> )
nodejs	function <b>set_rgbColor</b> ( <b>newval</b> )
php	function <b>set_rgbColor</b> ( <b>\$newval</b> )
cpp	int <b>set_rgbColor</b> ( int <b>newval</b> )
m	-(int) setRgbColor : (int) <b>newval</b>
pas	function <b>set_rgbColor</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_rgbColor</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_rgbColor</b> ( int <b>newval</b> )
java	int <b>set_rgbColor</b> ( int <b>newval</b> )
py	def <b>set_rgbColor</b> ( <b>newval</b> )
cmd	YColorLed <b>target set_rgbColor newval</b>

Encoding is done as follows: 0xRRGGBB.

**Parameters :**

**newval** an integer corresponding to the current color of the led, using a RGB color

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**colorled**→**set\_rgbColorAtPowerOn()**  
**colorled**→**setRgbColorAtPowerOn()**  
**colorled.set\_rgbColorAtPowerOn()**

**YColorLed**

Changes the color that the led will display by default when the module is turned on.

js	function <b>set_rgbColorAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_rgbColorAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_rgbColorAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_rgbColorAtPowerOn</b> ( int <b>newval</b> )
m	-(int) <b>setRgbColorAtPowerOn</b> : (int) <b>newval</b>
pas	function <b>set_rgbColorAtPowerOn</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_rgbColorAtPowerOn</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_rgbColorAtPowerOn</b> ( int <b>newval</b> )
java	int <b>set_rgbColorAtPowerOn</b> ( int <b>newval</b> )
py	def <b>set_rgbColorAtPowerOn</b> ( <b>newval</b> )
cmd	<b>YColorLed target set_rgbColorAtPowerOn newval</b>

This color will be displayed as soon as the module is powered on. Remember to call the `saveToFlash()` method of the module if the change should be kept.

#### Parameters :

**newval** an integer corresponding to the color that the led will display by default when the module is turned on

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**colored**→**set\_userdata()****YColorLed****colored**→**setUserData()****colored.set\_userdata()**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**colorled→wait\_async()****YColorLed**

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.

## 3.6. Compass function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindCompass(func)

Retrieves a compass for a given identifier.

#### yFirstCompass()

Starts the enumeration of compasses currently accessible.

### YCompass methods

#### compass→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### compass→describe()

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

#### compass→get\_advertisedValue()

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

#### compass→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### compass→get\_currentValue()

Returns the current value of the relative bearing.

#### compass→get\_errorMessage()

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

#### compass→get\_errorType()

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

#### compass→get\_friendlyName()

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

#### compass→get\_functionDescriptor()

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

#### compass→get\_functionId()

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

#### compass→get\_hardwareId()

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

**compass→get\_highestValue()**

Returns the maximal value observed for the relative bearing since the device was started.

**compass→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**compass→get\_logicalName()**

Returns the logical name of the compass.

**compass→get\_lowestValue()**

Returns the minimal value observed for the relative bearing since the device was started.

**compass→get\_magneticHeading()**

Returns the magnetic heading, regardless of the configured bearing.

**compass→get\_module()**

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

**compass→get\_module\_async(callback, context)**

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

**compass→get\_recordedData(startTime, endTime)**

Retrieves a `DataSet` object holding historical data for this sensor, for a specified time interval.

**compass→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**compass→get\_resolution()**

Returns the resolution of the measured values.

**compass→get\_unit()**

Returns the measuring unit for the relative bearing.

**compass→get\_userData()**

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

**compass→isOnline()**

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

**compass→isOnline\_async(callback, context)**

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

**compass→load(msValidity)**

Preloads the compass cache with a specified validity duration.

**compass→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

**compass→load\_async(msValidity, callback, context)**

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

**compass→nextCompass()**

Continues the enumeration of compasses started using `yFirstCompass()`.

**compass→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**compass→registerValueCallback(callback)**

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

**compass→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**compass→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

### 3. Reference

---

**compass**→**set\_logicalName**(newval)

Changes the logical name of the compass.

**compass**→**set\_lowestValue**(newval)

Changes the recorded minimal value observed.

**compass**→**set\_reportFrequency**(newval)

Changes the timed value notification frequency for this function.

**compass**→**set\_resolution**(newval)

Changes the resolution of the measured physical values.

**compass**→**set\_userData**(data)

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

**compass**→**wait\_async**(callback, context)

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

---

## YCompass.FindCompass() yFindCompass()YCompass.FindCompass()

## YCompass

Retrieves a compass for a given identifier.

js	function <b>yFindCompass</b> ( <b>func</b> )
nodejs	function <b>FindCompass</b> ( <b>func</b> )
php	function <b>yFindCompass</b> ( <b>\$func</b> )
cpp	YCompass* <b>yFindCompass</b> ( const string& <b>func</b> )
m	YCompass* <b>yFindCompass</b> ( NSString* <b>func</b> )
pas	function <b>yFindCompass</b> ( <b>func</b> : string): TYCompass
vb	function <b>yFindCompass</b> ( ByVal <b>func</b> As String) As YCompass
cs	YCompass <b>FindCompass</b> ( string <b>func</b> )
java	YCompass <b>FindCompass</b> ( String <b>func</b> )
py	def <b>FindCompass</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the compass is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YCompass.isOnline()` to test if the compass is indeed online at a given time. In case of ambiguity when looking for a compass 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 compass

### Returns :

a `YCompass` object allowing you to drive the compass.

**YCompass.FirstCompass()****YCompass****yFirstCompass()YCompass.FirstCompass()**

Starts the enumeration of compasses currently accessible.

js	function <b>yFirstCompass</b> ( )
nodejs	function <b>FirstCompass</b> ( )
php	function <b>yFirstCompass</b> ( )
cpp	YCompass* <b>yFirstCompass</b> ( )
m	YCompass* <b>yFirstCompass</b> ( )
pas	function <b>yFirstCompass</b> ( ): TYCompass
vb	function <b>yFirstCompass</b> ( ) As YCompass
cs	YCompass <b>FirstCompass</b> ( )
java	YCompass <b>FirstCompass</b> ( )
py	def <b>FirstCompass</b> ( )

Use the method `YCompass.nextCompass( )` to iterate on next compasses.

**Returns :**

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



## compass→calibrateFromPoints() compass.calibrateFromPoints()

YCompass

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

js	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
nodejs	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
php	function <b>calibrateFromPoints</b> ( <b>\$rawValues</b> , <b>\$refValues</b> )
cpp	int <b>calibrateFromPoints</b> ( vector<double> <b>rawValues</b> , vector<double> <b>refValues</b> )
m	-(int) <b>calibrateFromPoints</b> : (NSMutableArray*) <b>rawValues</b> : (NSMutableArray*) <b>refValues</b>
pas	function <b>calibrateFromPoints</b> ( <b>rawValues</b> : TDoubleArray, <b>refValues</b> : TDoubleArray): LongInt
vb	procedure <b>calibrateFromPoints</b> ( )
cs	int <b>calibrateFromPoints</b> ( List<double> <b>rawValues</b> , List<double> <b>refValues</b> )
java	int <b>calibrateFromPoints</b> ( ArrayList<Double> <b>rawValues</b> , ArrayList<Double> <b>refValues</b> )
py	def <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
cmd	YCompass <b>target</b> <b>calibrateFromPoints</b> <b>rawValues</b> <b>refValues</b>

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**compass→describe()****compass.describe()****YCompass**

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

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

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(BadCustomName.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 compass (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**compass**→**get\_advertisedValue()****YCompass****compass**→**advertisedValue()****compass.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the compass (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**compass**→**get\_currentRawValue()**  
**compass**→**currentRawValue()**  
**compass.get\_currentRawValue()**

**YCompass**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YCompass <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**compass**→**get\_currentValue()****YCompass****compass**→**currentValue()****compass.get\_currentValue()**

Returns the current value of the relative bearing.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YCompass <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the relative bearing

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**compass**→**get\_errorMessage()****YCompass****compass**→**errorMessage()****compass.get\_errorMessage()**

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

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the compass object

**compass**→**get\_errorType()****YCompass****compass**→**errorType()****compass.get\_errorType()**

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

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the compass object

**compass**→**get\_friendlyName()****compass**→**friendlyName()****compass.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the compass using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.



**compass**→**get\_functionDescriptor()**  
**compass**→**functionDescriptor()**  
**compass.get\_functionDescriptor()**

**YCompass**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**compass**→**get\_functionId()****YCompass****compass**→**functionId()****compass.get\_functionId()**

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

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

For example `relay1`

**Returns :**

a string that identifies the compass (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**compass**→**get\_hardwareId()****YCompass****compass**→**hardwareId()****compass.get\_hardwareId()**

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

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

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

**Returns :**

a string that uniquely identifies the compass (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**compass**→**get\_highestValue()****YCompass****compass**→**highestValue()****compass.get\_highestValue()**

Returns the maximal value observed for the relative bearing since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YCompass <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the relative bearing since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**compass→get\_logFrequency()****YCompass****compass→logFrequency()****compass.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
c++	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YCompass <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**compass**→**get\_logicalName()****YCompass****compass**→**logicalName()****compass.get\_logicalName()**

Returns the logical name of the compass.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YCompass <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the compass. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**compass**→**get\_lowestValue()****YCompass****compass**→**lowestValue()****compass.get\_lowestValue()**

Returns the minimal value observed for the relative bearing since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YCompass <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the relative bearing since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**compass**→**get\_magneticHeading()****YCompass****compass**→**magneticHeading()****compass.get\_magneticHeading()**

Returns the magnetic heading, regardless of the configured bearing.

js	function <b>get_magneticHeading</b> ( )
nodejs	function <b>get_magneticHeading</b> ( )
php	function <b>get_magneticHeading</b> ( )
cpp	double <b>get_magneticHeading</b> ( )
m	-(double) magneticHeading
pas	function <b>get_magneticHeading</b> ( ): double
vb	function <b>get_magneticHeading</b> ( ) As Double
cs	double <b>get_magneticHeading</b> ( )
java	double <b>get_magneticHeading</b> ( )
py	def <b>get_magneticHeading</b> ( )
cmd	YCompass <b>target</b> <b>get_magneticHeading</b>

**Returns :**

a floating point number corresponding to the magnetic heading, regardless of the configured bearing

On failure, throws an exception or returns Y\_MAGNETICHEADING\_INVALID.



**compass**→**get\_module()****YCompass****compass**→**module()****compass.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	<code>YModule *</code> <b>get_module</b> ( )
m	-( <code>YModule*</code> ) module
pas	function <b>get_module</b> ( ): <code>TYModule</code>
vb	function <b>get_module</b> ( ) As <code>YModule</code>
cs	<code>YModule</code> <b>get_module</b> ( )
java	<code>YModule</code> <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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`

**compass**→**get\_module\_async()****YCompass****compass**→**module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**compass→get\_recordedData()****YCompass****compass→recordedData()****compass.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YCompass <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**compass→get\_reportFrequency()****YCompass****compass→reportFrequency()****compass.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YCompass <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**compass**→**get\_resolution()****YCompass****compass**→**resolution()****compass.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YCompass <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**compass**→**get\_unit()****compass**→**unit()****compass.get\_unit()**

Returns the measuring unit for the relative bearing.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YCompass <b>target get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the relative bearing

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**compass**→**get\_userData()****YCompass****compass**→**userData()****compass.get\_userData()**

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

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

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.

**compass**→**isOnline()****compass.isOnline()****YCompass**

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

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

If there is a cached value for the compass 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 compass.

**Returns :**

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



**compass**→**isOnline\_async()****YCompass**

Checks if the compass 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 compass 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.

**compass**→**load()****compass.load()****YCompass**

Preloads the compass cache with a specified validity duration.

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

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 traffic 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.

## compass→loadCalibrationPoints() compass.loadCalibrationPoints()

YCompass

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YCompass target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**compass→load\_async()****YCompass**

Preloads the compass 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 traffic 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.

**compass**→**nextCompass()****compass.nextCompass()****YCompass**

Continues the enumeration of compasses started using `yFirstCompass()`.

js	function <b>nextCompass</b> ( )
nodejs	function <b>nextCompass</b> ( )
php	function <b>nextCompass</b> ( )
cpp	YCompass * <b>nextCompass</b> ( )
m	-(YCompass*) <b>nextCompass</b>
pas	function <b>nextCompass</b> ( ): TYCompass
vb	function <b>nextCompass</b> ( ) As YCompass
cs	YCompass <b>nextCompass</b> ( )
java	YCompass <b>nextCompass</b> ( )
py	def <b>nextCompass</b> ( )

**Returns :**

a pointer to a `YCompass` object, corresponding to a compass currently online, or a `null` pointer if there are no more compasses to enumerate.

## compass→registerTimedReportCallback() compass.registerTimedReportCallback()

YCompass

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
c++	int <b>registerTimedReportCallback</b> ( YCompassTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YCompassTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYCompassTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## compass→registerValueCallback() compass.registerValueCallback()

## YCompass

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YCompassValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YCompassValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYCompassValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**compass**→**set\_highestValue()****YCompass****compass**→**setHighestValue()****compass.set\_highestValue()**

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YCompass <b>target set_highestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**compass→set\_logFrequency()**  
**compass→setLogFrequency()**  
**compass.set\_logFrequency()**

**YCompass**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YCompass <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the datalogger recording frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**compass**→**set\_logicalName()****compass**→**setLogicalName()****compass.set\_logicalName()**

Changes the logical name of the compass.

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

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 compass.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**compass**→**set\_lowestValue()****YCompass****compass**→**setLowestValue()****compass.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YCompass <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**compass→set\_reportFrequency()****compass→setReportFrequency()****compass.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YCompass <b>target set_reportFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**compass**→**set\_resolution()****YCompass****compass**→**setResolution()****compass.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YCompass <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**compass**→**set\_userData()****compass**→**setUserData()****compass.set\_userData()**

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

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

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

**compass**→**wait\_async()****YCompass**

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.

## 3.7. Current function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindCurrent(func)

Retrieves a current sensor for a given identifier.

#### yFirstCurrent()

Starts the enumeration of current sensors currently accessible.

### YCurrent methods

#### current→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### current→describe()

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

#### current→get\_advertisedValue()

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

#### current→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### current→get\_currentValue()

Returns the current measure for the current.

#### current→get\_errorMessage()

Returns the error message of the latest error with the current sensor.

#### current→get\_errorType()

Returns the numerical error code of the latest error with the current sensor.

#### current→get\_friendlyName()

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

#### current→get\_functionDescriptor()

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

#### current→get\_functionId()

Returns the hardware identifier of the current sensor, without reference to the module.

#### current→get\_hardwareId()

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



**current→get\_highestValue()**

Returns the maximal value observed for the current.

**current→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**current→get\_logicalName()**

Returns the logical name of the current sensor.

**current→get\_lowestValue()**

Returns the minimal value observed for the current.

**current→get\_module()**

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

**current→get\_module\_async(callback, context)**

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

**current→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**current→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**current→get\_resolution()**

Returns the resolution of the measured values.

**current→get\_unit()**

Returns the measuring unit for the current.

**current→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**current→isOnline()**

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

**current→isOnline\_async(callback, context)**

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

**current→load(msValidity)**

Preloads the current sensor cache with a specified validity duration.

**current→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**current→load\_async(msValidity, callback, context)**

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

**current→nextCurrent()**

Continues the enumeration of current sensors started using yFirstCurrent ( ).

**current→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**current→registerValueCallback(callback)**

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

**current→set\_highestValue(newval)**

Changes the recorded maximal value observed pour the current.

**current→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**current→set\_logicalName(newval)**

Changes the logical name of the current sensor.

### 3. Reference

---

**current**→**set\_lowestValue**(newval)

Changes the recorded minimal value observed pour the current.

**current**→**set\_reportFrequency**(newval)

Changes the timed value notification frequency for this function.

**current**→**set\_resolution**(newval)

Changes the resolution of the measured values.

**current**→**set\_userData**(data)

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

**current**→**wait\_async**(callback, context)

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

## YCurrent.FindCurrent() yFindCurrent()YCurrent.FindCurrent()

YCurrent

Retrieves a current sensor for a given identifier.

js	function <b>yFindCurrent</b> ( <b>func</b> )
nodejs	function <b>FindCurrent</b> ( <b>func</b> )
php	function <b>yFindCurrent</b> ( <b>\$func</b> )
cpp	YCurrent* <b>yFindCurrent</b> ( const string& <b>func</b> )
m	YCurrent* <b>yFindCurrent</b> ( NSString* <b>func</b> )
pas	function <b>yFindCurrent</b> ( <b>func</b> : string): TYCurrent
vb	function <b>yFindCurrent</b> ( ByVal <b>func</b> As String) As YCurrent
cs	YCurrent <b>FindCurrent</b> ( string <b>func</b> )
java	YCurrent <b>FindCurrent</b> ( String <b>func</b> )
py	def <b>FindCurrent</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the current sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YCurrent.isOnline()` to test if the current sensor is indeed online at a given time. In case of ambiguity when looking for a current sensor 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 current sensor

### Returns :

a `YCurrent` object allowing you to drive the current sensor.

**YCurrent.FirstCurrent()****YCurrent****yFirstCurrent()****YCurrent.FirstCurrent()**

Starts the enumeration of current sensors currently accessible.

js	function <b>yFirstCurrent</b> ( )
nodejs	function <b>FirstCurrent</b> ( )
php	function <b>yFirstCurrent</b> ( )
cpp	YCurrent* <b>yFirstCurrent</b> ( )
m	YCurrent* <b>yFirstCurrent</b> ( )
pas	function <b>yFirstCurrent</b> ( ): TYCurrent
vb	function <b>yFirstCurrent</b> ( ) As YCurrent
cs	YCurrent <b>FirstCurrent</b> ( )
java	YCurrent <b>FirstCurrent</b> ( )
py	def <b>FirstCurrent</b> ( )

Use the method `YCurrent.nextCurrent( )` to iterate on next current sensors.

**Returns :**

a pointer to a `YCurrent` object, corresponding to the first current sensor currently online, or a `null` pointer if there are none.

## current→calibrateFromPoints() current.calibrateFromPoints()

YCurrent

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js      function calibrateFromPoints( rawValues, refValues)
node.js function calibrateFromPoints( rawValues, refValues)
php     function calibrateFromPoints( $rawValues, $refValues)
cpp     int calibrateFromPoints( vector<double> rawValues,
                                vector<double> refValues)

m       -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function calibrateFromPoints( rawValues: TDoubleArray,
                                refValues: TDoubleArray): LongInt

vb      procedure calibrateFromPoints( )

cs      int calibrateFromPoints( List<double> rawValues,
                                List<double> refValues)

java    int calibrateFromPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def calibrateFromPoints( rawValues, refValues)

cmd     YCurrent target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**current→describe()****current.describe()****YCurrent**

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

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

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 current sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**current**→**get\_advertisedValue()****YCurrent****current**→**advertisedValue()****current.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the current sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**current**→**get\_currentRawValue()****YCurrent****current**→**currentRawValue()****current.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

<code>js</code>	<code>function <b>get_currentRawValue</b>( )</code>
<code>nodejs</code>	<code>function <b>get_currentRawValue</b>( )</code>
<code>php</code>	<code>function <b>get_currentRawValue</b>( )</code>
<code>cpp</code>	<code>double <b>get_currentRawValue</b>( )</code>
<code>m</code>	<code>-(double) currentRawValue</code>
<code>pas</code>	<code>function <b>get_currentRawValue</b>( ): double</code>
<code>vb</code>	<code>function <b>get_currentRawValue</b>( ) As Double</code>
<code>cs</code>	<code>double <b>get_currentRawValue</b>( )</code>
<code>java</code>	<code>double <b>get_currentRawValue</b>( )</code>
<code>py</code>	<code>def <b>get_currentRawValue</b>( )</code>
<code>cmd</code>	<code>YCurrent <b>target</b> <b>get_currentRawValue</b></code>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.



**current**→**get\_currentValue()****YCurrent****current**→**currentValue()****current.get\_currentValue()**

Returns the current measure for the current.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) <b>currentValue</b>
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YCurrent <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the current

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**current**→**get\_errorMessage()****YCurrent****current**→**errorMessage()****current.get\_errorMessage()**

Returns the error message of the latest error with the current sensor.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the current sensor object

**current**→**get\_errorType()****YCurrent****current**→**errorType()****current.get\_errorType()**

Returns the numerical error code of the latest error with the current sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the current sensor object

**current**→**get\_friendlyName()****YCurrent****current**→**friendlyName()****current.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
c++	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

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

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

**current→get\_functionDescriptor()****YCurrent****current→functionDescriptor()****current.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**current**→**get\_functionId()****YCurrent****current**→**functionId()****current.get\_functionId()**

Returns the hardware identifier of the current sensor, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the current sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**current**→**get\_hardwareId()****YCurrent****current**→**hardwareId()****current.get\_hardwareId()**

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

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

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

**Returns :**

a string that uniquely identifies the current sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**current**→**get\_highestValue()****YCurrent****current**→**highestValue()****current.get\_highestValue()**

Returns the maximal value observed for the current.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YCurrent <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the current

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.



**current→get\_logFrequency()****YCurrent****current→logFrequency()current.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YCurrent <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**current**→**get\_logicalName()****YCurrent****current**→**logicalName()****current.get\_logicalName()**

Returns the logical name of the current sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YCurrent <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the current sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**current**→**get\_lowestValue()****YCurrent****current**→**lowestValue()****current.get\_lowestValue()**

Returns the minimal value observed for the current.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YCurrent <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the current

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**current**→**get\_module()****YCurrent****current**→**module()****current.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**current**→**get\_module\_async()**  
**current**→**module\_async()**

**YCurrent**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**current**→**get\_recordedData()****YCurrent****current**→**recordedData()****current.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YCurrent <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**current→get\_reportFrequency()****YCurrent****current→reportFrequency()****current.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YCurrent <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**current**→**get\_resolution()****YCurrent****current**→**resolution()****current.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YCurrent <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.



**current**→**get\_unit()****YCurrent****current**→**unit()****current.get\_unit()**

Returns the measuring unit for the current.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YCurrent <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the current

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**current**→**get\_userData()****YCurrent****current**→**userData()****current.get\_userData()**

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

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

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.

**current→isOnline()current.isOnline()****YCurrent**

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

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

If there is a cached value for the current sensor 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 current sensor.

**Returns :**

`true` if the current sensor can be reached, and `false` otherwise

**current→isOnline\_async()****YCurrent**

Checks if the current sensor 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 current sensor 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.

**current→load()****current.load()****YCurrent**

Preloads the current sensor cache with a specified validity duration.

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

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 traffic 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.

## current→loadCalibrationPoints() current.loadCalibrationPoints()

YCurrent

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YCurrent target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**current→load\_async()****YCurrent**

Preloads the current sensor 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 traffic 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.

**current**→**nextCurrent()****current.nextCurrent()****YCurrent**

Continues the enumeration of current sensors started using `yFirstCurrent()`.

js	function <b>nextCurrent</b> ( )
nodejs	function <b>nextCurrent</b> ( )
php	function <b>nextCurrent</b> ( )
cpp	YCurrent * <b>nextCurrent</b> ( )
m	-(YCurrent*) <b>nextCurrent</b>
pas	function <b>nextCurrent</b> ( ): TYCurrent
vb	function <b>nextCurrent</b> ( ) As YCurrent
cs	YCurrent <b>nextCurrent</b> ( )
java	YCurrent <b>nextCurrent</b> ( )
py	def <b>nextCurrent</b> ( )

**Returns :**

a pointer to a `YCurrent` object, corresponding to a current sensor currently online, or a `null` pointer if there are no more current sensors to enumerate.



## current→registerTimedReportCallback() current.registerTimedReportCallback()

YCurrent

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YCurrentTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YCurrentTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYCurrentTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## current→registerValueCallback() current.registerValueCallback()

YCurrent

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YCurrentValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YCurrentValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYCurrentValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**current**→**set\_highestValue()****YCurrent****current**→**setHighestValue()****current.set\_highestValue()**

Changes the recorded maximal value observed pour the current.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YCurrent <b>target set_highestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed pour the current

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**current**→**set\_logFrequency()****YCurrent****current**→**setLogFrequency()****current.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YCurrent <b>target</b> <b>set_logFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**current**→**set\_logicalName()****YCurrent****current**→**setLogicalName()****current.set\_logicalName()**

Changes the logical name of the current sensor.

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

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 current sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**current**→**set\_lowestValue()****YCurrent****current**→**setLowestValue()****current.set\_lowestValue()**

Changes the recorded minimal value observed pour the current.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YCurrent <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed pour the current

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**current→set\_reportFrequency()**  
**current→setReportFrequency()**  
**current.set\_reportFrequency()**

**YCurrent**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YCurrent <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**current**→**set\_resolution()****YCurrent****current**→**setResolution()****current.set\_resolution()**

Changes the resolution of the measured values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YCurrent <b>target</b> <b>set_resolution</b> <b>newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**current**→**set\_userData()****YCurrent****current**→**setUserData()****current.set\_userData()**

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

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

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

**current→wait\_async()****YCurrent**

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.

## 3.8. DataLogger function interface

Yoctopuce sensors include a non-volatile memory capable of storing ongoing measured data automatically, without requiring a permanent connection to a computer. The DataLogger function controls the global parameters of the internal data logger.

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

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

### Global functions

#### yFindDataLogger(func)

Retrieves a data logger for a given identifier.

#### yFirstDataLogger()

Starts the enumeration of data loggers currently accessible.

### YDataLogger methods

#### datalogger→describe()

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

#### datalogger→forgetAllDataStreams()

Clears the data logger memory and discards all recorded data streams.

#### datalogger→get\_advertisedValue()

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

#### datalogger→get\_autoStart()

Returns the default activation state of the data logger on power up.

#### datalogger→get\_currentRunIndex()

Returns the current run number, corresponding to the number of times the module was powered on with the dataLogger enabled at some point.

#### datalogger→get\_dataSets()

Returns a list of YDataSet objects that can be used to retrieve all measures stored by the data logger.

#### datalogger→get\_dataStreams(v)

Builds a list of all data streams hold by the data logger (legacy method).

#### datalogger→get\_errorMessage()

Returns the error message of the latest error with the data logger.

#### datalogger→get\_errorType()

Returns the numerical error code of the latest error with the data logger.

#### datalogger→get\_friendlyName()

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

#### datalogger→get\_functionDescriptor()

	Returns a unique identifier of type YFUN_DESCR corresponding to the function.
<b>datalogger→get_functionId()</b>	Returns the hardware identifier of the data logger, without reference to the module.
<b>datalogger→get_hardwareId()</b>	Returns the unique hardware identifier of the data logger in the form SERIAL . FUNCTIONID.
<b>datalogger→get_logicalName()</b>	Returns the logical name of the data logger.
<b>datalogger→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>datalogger→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>datalogger→get_recording()</b>	Returns the current activation state of the data logger.
<b>datalogger→get_timeUTC()</b>	Returns the Unix timestamp for current UTC time, if known.
<b>datalogger→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set_userData.
<b>datalogger→isOnline()</b>	Checks if the data logger is currently reachable, without raising any error.
<b>datalogger→isOnline_async(callback, context)</b>	Checks if the data logger is currently reachable, without raising any error (asynchronous version).
<b>datalogger→load(msValidity)</b>	Preloads the data logger cache with a specified validity duration.
<b>datalogger→load_async(msValidity, callback, context)</b>	Preloads the data logger cache with a specified validity duration (asynchronous version).
<b>datalogger→nextDataLogger()</b>	Continues the enumeration of data loggers started using yFirstDataLogger( ).
<b>datalogger→registerValueCallback(callback)</b>	Registers the callback function that is invoked on every change of advertised value.
<b>datalogger→set_autoStart(newval)</b>	Changes the default activation state of the data logger on power up.
<b>datalogger→set_logicalName(newval)</b>	Changes the logical name of the data logger.
<b>datalogger→set_recording(newval)</b>	Changes the activation state of the data logger to start/stop recording data.
<b>datalogger→set_timeUTC(newval)</b>	Changes the current UTC time reference used for recorded data.
<b>datalogger→set_userData(data)</b>	Stores a user context provided as argument in the userData attribute of the function.
<b>datalogger→wait_async(callback, context)</b>	Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YDataLogger.FindDataLogger()****YDataLogger****yFindDataLogger()YDataLogger.FindDataLogger()**

Retrieves a data logger for a given identifier.

js	function <b>yFindDataLogger</b> ( <b>func</b> )
nodejs	function <b>FindDataLogger</b> ( <b>func</b> )
php	function <b>yFindDataLogger</b> ( <b>\$func</b> )
cpp	YDataLogger* <b>yFindDataLogger</b> ( string <b>func</b> )
m	+(YDataLogger*) <b>yFindDataLogger</b> : (NSString*) <b>func</b>
pas	function <b>yFindDataLogger</b> ( <b>func</b> : string): TYDataLogger
vb	function <b>yFindDataLogger</b> ( ByVal <b>func</b> As String) As YDataLogger
cs	YDataLogger <b>FindDataLogger</b> ( string <b>func</b> )
java	YDataLogger <b>FindDataLogger</b> ( String <b>func</b> )
py	def <b>FindDataLogger</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the data logger is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YDataLogger.isOnline()` to test if the data logger is indeed online at a given time. In case of ambiguity when looking for a data logger 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 data logger

**Returns :**

a `YDataLogger` object allowing you to drive the data logger.

**YDataLogger.FirstDataLogger()****YDataLogger****yFirstDataLogger()YDataLogger.FirstDataLogger()**

Starts the enumeration of data loggers currently accessible.

js	function <b>yFirstDataLogger</b> ( )
nodejs	function <b>FirstDataLogger</b> ( )
php	function <b>yFirstDataLogger</b> ( )
cpp	YDataLogger* <b>yFirstDataLogger</b> ( )
m	YDataLogger* <b>yFirstDataLogger</b> ( )
pas	function <b>yFirstDataLogger</b> ( ): TYDataLogger
vb	function <b>yFirstDataLogger</b> ( ) As YDataLogger
cs	YDataLogger <b>FirstDataLogger</b> ( )
java	YDataLogger <b>FirstDataLogger</b> ( )
py	def <b>FirstDataLogger</b> ( )

Use the method `YDataLogger.nextDataLogger( )` to iterate on next data loggers.

**Returns :**

a pointer to a `YDataLogger` object, corresponding to the first data logger currently online, or a `null` pointer if there are none.

**datalogger→describe()****datalogger.describe()****YDataLogger**

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

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

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(BadCustomName.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 data logger (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**datalogger→forgetAllDataStreams()  
datalogger.forgetAllDataStreams()****YDataLogger**

Clears the data logger memory and discards all recorded data streams.

js	function <b>forgetAllDataStreams</b> ( )
nodejs	function <b>forgetAllDataStreams</b> ( )
php	function <b>forgetAllDataStreams</b> ( )
cpp	int <b>forgetAllDataStreams</b> ( )
m	-(int) <b>forgetAllDataStreams</b>
pas	function <b>forgetAllDataStreams</b> ( ): LongInt
vb	function <b>forgetAllDataStreams</b> ( ) As Integer
cs	int <b>forgetAllDataStreams</b> ( )
java	int <b>forgetAllDataStreams</b> ( )
py	def <b>forgetAllDataStreams</b> ( )
cmd	YDataLogger <b>target</b> <b>forgetAllDataStreams</b>

This method also resets the current run index to zero.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**datalogger→get\_advertisedValue()****YDataLogger****datalogger→advertisedValue()****datalogger.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the data logger (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**datalogger**→**get\_autoStart()****YDataLogger****datalogger**→**autoStart()****datalogger.get\_autoStart()**

Returns the default activation state of the data logger on power up.

js	function <b>get_autoStart</b> ( )
nodejs	function <b>get_autoStart</b> ( )
php	function <b>get_autoStart</b> ( )
cpp	Y_AUTOSTART_enum <b>get_autoStart</b> ( )
m	-(Y_AUTOSTART_enum) autoStart
pas	function <b>get_autoStart</b> ( ): Integer
vb	function <b>get_autoStart</b> ( ) As Integer
cs	int <b>get_autoStart</b> ( )
java	int <b>get_autoStart</b> ( )
py	def <b>get_autoStart</b> ( )
cmd	YDataLogger <b>target</b> <b>get_autoStart</b>

**Returns :**

either Y\_AUTOSTART\_OFF or Y\_AUTOSTART\_ON, according to the default activation state of the data logger on power up

On failure, throws an exception or returns Y\_AUTOSTART\_INVALID.

**datalogger→get\_currentRunIndex()****YDataLogger****datalogger→currentRunIndex()****datalogger.get\_currentRunIndex()**

Returns the current run number, corresponding to the number of times the module was powered on with the dataLogger enabled at some point.

js	function <b>get_currentRunIndex</b> ( )
nodejs	function <b>get_currentRunIndex</b> ( )
php	function <b>get_currentRunIndex</b> ( )
cpp	int <b>get_currentRunIndex</b> ( )
m	-(int) currentRunIndex
pas	function <b>get_currentRunIndex</b> ( ): LongInt
vb	function <b>get_currentRunIndex</b> ( ) As Integer
cs	int <b>get_currentRunIndex</b> ( )
java	int <b>get_currentRunIndex</b> ( )
py	def <b>get_currentRunIndex</b> ( )
cmd	YDataLogger <b>target</b> <b>get_currentRunIndex</b>

**Returns :**

an integer corresponding to the current run number, corresponding to the number of times the module was powered on with the dataLogger enabled at some point

On failure, throws an exception or returns Y\_CURRENTRUNINDEX\_INVALID.

**datalogger**→**get\_dataSets()****YDataLogger****datalogger**→**dataSets()****datalogger.get\_dataSets()**

Returns a list of YDataSet objects that can be used to retrieve all measures stored by the data logger.

js	function <b>get_dataSets</b> ( )
nodejs	function <b>get_dataSets</b> ( )
php	function <b>get_dataSets</b> ( )
cpp	vector<YDataSet> <b>get_dataSets</b> ( )
m	-(NSMutableArray*) dataSets
pas	function <b>get_dataSets</b> ( ): TYDataSetArray
vb	function <b>get_dataSets</b> ( ) As List
cs	List<YDataSet> <b>get_dataSets</b> ( )
java	ArrayList<YDataSet> <b>get_dataSets</b> ( )
py	def <b>get_dataSets</b> ( )
cmd	YDataLogger <b>target</b> <b>get_dataSets</b>

This function only works if the device uses a recent firmware, as YDataSet objects are not supported by firmwares older than version 13000.

**Returns :**

a list of YDataSet object.

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

**datalogger→get\_dataStreams()**  
**datalogger→dataStreams()**  
**datalogger.get\_dataStreams()**

**YDataLogger**

Builds a list of all data streams hold by the data logger (legacy method).

js	function <b>get_dataStreams</b> ( <b>v</b> )
nodejs	function <b>get_dataStreams</b> ( <b>v</b> )
php	function <b>get_dataStreams</b> ( <b>&amp;\$v</b> )
cpp	int <b>get_dataStreams</b> ( )
m	-(int) dataStreams : (NSArray**) <b>v</b>
pas	function <b>get_dataStreams</b> ( <b>v</b> : Tlist): integer
vb	procedure <b>get_dataStreams</b> ( ByVal <b>v</b> As List)
cs	int <b>get_dataStreams</b> ( List<YDataStream> <b>v</b> )
java	int <b>get_dataStreams</b> ( ArrayList<YDataStream> <b>v</b> )
py	def <b>get_dataStreams</b> ( <b>v</b> )

The caller must pass by reference an empty array to hold YDataStream objects, and the function fills it with objects describing available data sequences.

This is the old way to retrieve data from the DataLogger. For new applications, you should rather use `get_dataSets()` method, or call directly `get_recordedData()` on the sensor object.

#### Parameters :

**v** an array of YDataStream objects to be filled in

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**datalogger→get\_errorMessage()****YDataLogger****datalogger→errorMessage()****datalogger.errorMessage()**

Returns the error message of the latest error with the data logger.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the data logger object

**datalogger**→**get\_errorType()****YDataLogger****datalogger**→**errorType()****datalogger.get\_errorType()**

Returns the numerical error code of the latest error with the data logger.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the data logger object

**datalogger→get\_friendlyName()****YDataLogger****datalogger→friendlyName()****datalogger.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the data logger using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.



**datalogger→get\_functionDescriptor()**  
**datalogger→functionDescriptor()**  
**datalogger.get\_functionDescriptor()**

**YDataLogger**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**datalogger**→**get\_functionId()****YDataLogger****datalogger**→**functionId()****datalogger.get\_functionId()**

Returns the hardware identifier of the data logger, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the data logger (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**datalogger→get\_hardwareId()**  
**datalogger→hardwareId()**  
**datalogger.get\_hardwareId()**

**YDataLogger**

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

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

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

**Returns :**

a string that uniquely identifies the data logger (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**datalogger→get\_logicalName()****YDataLogger****datalogger→logicalName()****datalogger.get\_logicalName()**

Returns the logical name of the data logger.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YDataLogger <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the data logger. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**datalogger→get\_module()****YDataLogger****datalogger→module()datalogger.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**datalogger**→**get\_module\_async()****YDataLogger****datalogger**→**module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**datalogger→get\_recording()****YDataLogger****datalogger→recording()datalogger.get\_recording()**

Returns the current activation state of the data logger.

js	function <b>get_recording</b> ( )
nodejs	function <b>get_recording</b> ( )
php	function <b>get_recording</b> ( )
cpp	Y_RECORDING_enum <b>get_recording</b> ( )
m	-(Y_RECORDING_enum) recording
pas	function <b>get_recording</b> ( ): Integer
vb	function <b>get_recording</b> ( ) As Integer
cs	int <b>get_recording</b> ( )
java	int <b>get_recording</b> ( )
py	def <b>get_recording</b> ( )
cmd	YDataLogger <b>target</b> <b>get_recording</b>

**Returns :**

either Y\_RECORDING\_OFF or Y\_RECORDING\_ON, according to the current activation state of the data logger

On failure, throws an exception or returns Y\_RECORDING\_INVALID.

**datalogger**→**get\_timeUTC()****YDataLogger****datalogger**→**timeUTC()****datalogger.get\_timeUTC()**

Returns the Unix timestamp for current UTC time, if known.

js	function <b>get_timeUTC</b> ( )
nodejs	function <b>get_timeUTC</b> ( )
php	function <b>get_timeUTC</b> ( )
cpp	s64 <b>get_timeUTC</b> ( )
m	-(s64) timeUTC
pas	function <b>get_timeUTC</b> ( ): int64
vb	function <b>get_timeUTC</b> ( ) As Long
cs	long <b>get_timeUTC</b> ( )
java	long <b>get_timeUTC</b> ( )
py	def <b>get_timeUTC</b> ( )
cmd	YDataLogger <b>target</b> <b>get_timeUTC</b>

**Returns :**

an integer corresponding to the Unix timestamp for current UTC time, if known

On failure, throws an exception or returns Y\_TIMEUTC\_INVALID.



**datalogger**→**get\_userdata()****YDataLogger****datalogger**→**userData()****datalogger.get\_userdata()**

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

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**datalogger**→**isOnline()****datalogger.isOnline()****YDataLogger**

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

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

If there is a cached value for the data logger 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 data logger.

**Returns :**

`true` if the data logger can be reached, and `false` otherwise

**datalogger→isOnline\_async()****YDataLogger**

Checks if the data logger 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 data logger 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.

**datalogger**→**load()****datalogger.load()****YDataLogger**

Preloads the data logger cache with a specified validity duration.

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

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 traffic 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.

**datalogger→load\_async()****YDataLogger**

Preloads the data logger 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 traffic 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.

**dataLogger**→**nextDataLogger()**  
**dataLogger.nextDataLogger()****YDataLogger**

Continues the enumeration of data loggers started using `yFirstDataLogger()`.

js	function <b>nextDataLogger</b> ( )
nodejs	function <b>nextDataLogger</b> ( )
php	function <b>nextDataLogger</b> ( )
c++	YDataLogger * <b>nextDataLogger</b> ( )
m	-(YDataLogger*) <b>nextDataLogger</b>
pas	function <b>nextDataLogger</b> ( ): TYDataLogger
vb	function <b>nextDataLogger</b> ( ) As YDataLogger
cs	YDataLogger <b>nextDataLogger</b> ( )
java	YDataLogger <b>nextDataLogger</b> ( )
py	def <b>nextDataLogger</b> ( )

**Returns :**

a pointer to a `YDataLogger` object, corresponding to a data logger currently online, or a `null` pointer if there are no more data loggers to enumerate.

## **datalogger→registerValueCallback()**

### **datalogger.registerValueCallback()**

## **YDataLogger**

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YDataLoggerValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YDataLoggerValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYDataLoggerValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**datalogger→set\_autoStart()****YDataLogger****datalogger→setAutoStart()datalogger.set\_autoStart()**

Changes the default activation state of the data logger on power up.

js	function <b>set_autoStart</b> ( <b>newval</b> )
nodejs	function <b>set_autoStart</b> ( <b>newval</b> )
php	function <b>set_autoStart</b> ( <b>\$newval</b> )
cpp	int <b>set_autoStart</b> ( Y_AUTOSTART_enum <b>newval</b> )
m	-(int) setAutoStart : (Y_AUTOSTART_enum) <b>newval</b>
pas	function <b>set_autoStart</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_autoStart</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_autoStart</b> ( int <b>newval</b> )
java	int <b>set_autoStart</b> ( int <b>newval</b> )
py	def <b>set_autoStart</b> ( <b>newval</b> )
cmd	YDataLogger <b>target set_autoStart newval</b>

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

**Parameters :**

**newval** either Y\_AUTOSTART\_OFF or Y\_AUTOSTART\_ON, according to the default activation state of the data logger on power up

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**datalogger→set\_logicalName()**  
**datalogger→setLogicalName()**  
**datalogger.set\_logicalName()**

**YDataLogger**

Changes the logical name of the data logger.

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

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 data logger.

#### Returns :

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**datalogger→set\_recording()****YDataLogger****datalogger→setRecording()****datalogger.set\_recording()**

Changes the activation state of the data logger to start/stop recording data.

js	function <b>set_recording</b> ( <b>newval</b> )
nodejs	function <b>set_recording</b> ( <b>newval</b> )
php	function <b>set_recording</b> ( <b>\$newval</b> )
cpp	int <b>set_recording</b> ( Y_RECORDING_enum <b>newval</b> )
m	-(int) setRecording : (Y_RECORDING_enum) <b>newval</b>
pas	function <b>set_recording</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_recording</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_recording</b> ( int <b>newval</b> )
java	int <b>set_recording</b> ( int <b>newval</b> )
py	def <b>set_recording</b> ( <b>newval</b> )
cmd	YDataLogger <b>target</b> <b>set_recording</b> <b>newval</b>

**Parameters :**

**newval** either Y\_RECORDING\_OFF or Y\_RECORDING\_ON, according to the activation state of the data logger to start/stop recording data

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**datalogger**→**set\_timeUTC()****YDataLogger****datalogger**→**setTimeUTC()****datalogger.set\_timeUTC()**

Changes the current UTC time reference used for recorded data.

js	function <b>set_timeUTC</b> ( <b>newval</b> )
nodejs	function <b>set_timeUTC</b> ( <b>newval</b> )
php	function <b>set_timeUTC</b> ( <b>\$newval</b> )
cpp	int <b>set_timeUTC</b> ( s64 <b>newval</b> )
m	-(int) setTimeUTC : (s64) <b>newval</b>
pas	function <b>set_timeUTC</b> ( <b>newval</b> : int64): integer
vb	function <b>set_timeUTC</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_timeUTC</b> ( long <b>newval</b> )
java	int <b>set_timeUTC</b> ( long <b>newval</b> )
py	def <b>set_timeUTC</b> ( <b>newval</b> )
cmd	YDataLogger <b>target set_timeUTC newval</b>

**Parameters :**

**newval** an integer corresponding to the current UTC time reference used for recorded data

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**datalogger**→**set\_userData()****YDataLogger****datalogger**→**setUserData()****datalogger.set\_userData()**

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

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

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

**datalogger**→**wait\_async()****YDataLogger**

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.

### 3.9. Formatted data sequence

A run is a continuous interval of time during which a module was powered on. A data run provides easy access to all data collected during a given run, providing on-the-fly resampling at the desired reporting rate.

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

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

YDataRun methods
<b>datarun→get_averageValue(measureName, pos)</b> Returns the average value of the measure observed at the specified time period.
<b>datarun→get_duration()</b> Returns the duration (in seconds) of the data run.
<b>datarun→get_maxValue(measureName, pos)</b> Returns the maximal value of the measure observed at the specified time period.
<b>datarun→get_measureNames()</b> Returns the names of the measures recorded by the data logger.
<b>datarun→get_minValue(measureName, pos)</b> Returns the minimal value of the measure observed at the specified time period.
<b>datarun→get_startTimeUTC()</b> Returns the start time of the data run, relative to the Jan 1, 1970.
<b>datarun→get_valueCount()</b> Returns the number of values accessible in this run, given the selected data samples interval.
<b>datarun→get_valueInterval()</b> Returns the number of seconds covered by each value in this run.
<b>datarun→set_valueInterval(valueInterval)</b> Changes the number of seconds covered by each value in this run.

**datarun→get\_averageValue()****YDataRun****datarun→averageValue()datarun.get\_averageValue()**

Returns the average value of the measure observed at the specified time period.

js	function <b>get_averageValue</b> ( <b>measureName</b> , <b>pos</b> )
nodejs	function <b>get_averageValue</b> ( <b>measureName</b> , <b>pos</b> )
php	function <b>get_averageValue</b> ( <b>\$measureName</b> , <b>\$pos</b> )
java	double <b>get_averageValue</b> ( String <b>measureName</b> , int <b>pos</b> )
py	def <b>get_averageValue</b> ( <b>measureName</b> , <b>pos</b> )

**Parameters :**

**measureName** the name of the desired measure (one of the names returned by `get_measureNames`)

**pos** the position index, between 0 and the value returned by `get_valueCount`

**Returns :**

a floating point number (the average value)

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

**datarun**→**get\_duration()**

**YDataRun**

**datarun**→**duration()****datarun.get\_duration()**

---

Returns the duration (in seconds) of the data run.

```
js function get_duration( )
```

```
nodejs function get_duration( )
```

```
php function get_duration( )
```

```
java long get_duration( )
```

```
py def get_duration( )
```

When the datalogger is actively recording and the specified run is the current run, calling this method reloads last sequence(s) from device to make sure it includes the latest recorded data.

**Returns :**

an unsigned number corresponding to the number of seconds between the beginning of the run (when the module was powered up) and the last recorded measure.



**datarun**→**get\_maxValue()****YDataRun****datarun**→**maxValue()****datarun.get\_maxValue()**

Returns the maximal value of the measure observed at the specified time period.

<code>js</code>	<code>function get_maxValue( measureName, pos)</code>
<code>nodejs</code>	<code>function get_maxValue( measureName, pos)</code>
<code>php</code>	<code>function get_maxValue( \$measureName, \$pos)</code>
<code>java</code>	<code>double get_maxValue( String measureName, int pos)</code>
<code>py</code>	<code>def get_maxValue( measureName, pos)</code>

**Parameters :**

**measureName** the name of the desired measure (one of the names returned by `get_measureNames`)

**pos** the position index, between 0 and the value returned by `get_valueCount`

**Returns :**

a floating point number (the maximal value)

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

**datarun→get\_measureNames()**

**YDataRun**

**datarun→measureNames()**

**datarun.get\_measureNames()**

---

Returns the names of the measures recorded by the data logger.

```
js function get_measureNames( )
nodejs function get_measureNames( )
php function get_measureNames( )
java ArrayList<String> get_measureNames( )
py def get_measureNames( )
```

In most case, the measure names match the hardware identifier of the sensor that produced the data.

**Returns :**

a list of strings (the measure names) On failure, throws an exception or returns an empty array.

**datarun→get\_minValue()****YDataRun****datarun→minValue()****datarun.get\_minValue()**

Returns the minimal value of the measure observed at the specified time period.

<code>js</code>	<code>function get_minValue( measureName, pos)</code>
<code>nodejs</code>	<code>function get_minValue( measureName, pos)</code>
<code>php</code>	<code>function get_minValue( \$measureName, \$pos)</code>
<code>java</code>	<code>double get_minValue( String measureName, int pos)</code>
<code>py</code>	<code>def get_minValue( measureName, pos)</code>

**Parameters :**

**measureName** the name of the desired measure (one of the names returned by `get_measureNames`)

**pos** the position index, between 0 and the value returned by `get_valueCount`

**Returns :**

a floating point number (the minimal value)

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

**datarun→get\_startTimeUTC()**

**YDataRun**

**datarun→startTimeUTC()**

---

Returns the start time of the data run, relative to the Jan 1, 1970.

If the UTC time was not set in the datalogger at any time during the recording of this data run, and if this is not the current run, this method returns 0.

**Returns :**

an unsigned number corresponding to the number of seconds between the Jan 1, 1970 and the beginning of this data run (i.e. Unix time representation of the absolute time).

**datarun→get\_valueCount()****YDataRun****datarun→valueCount()datarun.get\_valueCount()**

Returns the number of values accessible in this run, given the selected data samples interval.

js	function <b>get_valueCount</b> ( )
nodejs	function <b>get_valueCount</b> ( )
php	function <b>get_valueCount</b> ( )
java	int <b>get_valueCount</b> ( )
py	def <b>get_valueCount</b> ( )

When the datalogger is actively recording and the specified run is the current run, calling this method reloads last sequence(s) from device to make sure it includes the latest recorded data.

**Returns :**

an unsigned number corresponding to the run duration divided by the samples interval.

**datarun**→**get\_valueInterval()**

**YDataRun**

**datarun**→**valueInterval()****datarun.get\_valueInterval()**

---

Returns the number of seconds covered by each value in this run.

```
js function get_valueInterval( )
```

```
nodejs function get_valueInterval( )
```

```
php function get_valueInterval( )
```

```
java int get_valueInterval( )
```

```
py def get_valueInterval( )
```

By default, the value interval is set to the coarsest data rate archived in the data logger flash for this run. The value interval can however be configured at will to a different rate when desired.

**Returns :**

an unsigned number corresponding to a number of seconds covered by each data sample in the Run.

---

**datarun→set\_valueInterval()****YDataRun****datarun→setValueInterval()****datarun.set\_valueInterval()**

---

Changes the number of seconds covered by each value in this run.

```
js function set_valueInterval( valueInterval)
nodejs function set_valueInterval( valueInterval)
php function set_valueInterval( $valueInterval)
java void set_valueInterval( int valueInterval)
py def set_valueInterval( valueInterval)
```

By default, the value interval is set to the coarsest data rate archived in the data logger flash for this run. The value interval can however be configured at will to a different rate when desired.

**Parameters :**

**valueInterval** an integer number of seconds.

**Returns :**

nothing

## 3.10. Recorded data sequence

YDataSet objects make it possible to retrieve a set of recorded measures for a given sensor and a specified time interval. They can be used to load data points with a progress report. When the YDataSet object is instantiated by the `get_recordedData()` function, no data is yet loaded from the module. It is only when the `loadMore()` method is called over and over than data will be effectively loaded from the dataLogger.

A preview of available measures is available using the function `get_preview()` as soon as `loadMore()` has been called once. Measures themselves are available using function `get_measures()` when loaded by subsequent calls to `loadMore()`.

This class can only be used on devices that use a recent firmware, as YDataSet objects are not supported by firmwares older than version 13000.

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

js	<code>&lt;script type='text/javascript' src='yocto_api.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YAPI = yoctolib.YAPI;</code> <code>var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

### YDataSet methods

#### **dataset→get\_endTimeUTC()**

Returns the end time of the dataset, relative to the Jan 1, 1970.

#### **dataset→get\_functionId()**

Returns the hardware identifier of the function that performed the measure, without reference to the module.

#### **dataset→get\_hardwareId()**

Returns the unique hardware identifier of the function who performed the measures, in the form `SERIAL.FUNCTIONID`.

#### **dataset→get\_measures()**

Returns all measured values currently available for this DataSet, as a list of YMeasure objects.

#### **dataset→get\_preview()**

Returns a condensed version of the measures that can retrieved in this YDataSet, as a list of YMeasure objects.

#### **dataset→get\_progress()**

Returns the progress of the downloads of the measures from the data logger, on a scale from 0 to 100.

#### **dataset→get\_startTimeUTC()**

Returns the start time of the dataset, relative to the Jan 1, 1970.

#### **dataset→get\_summary()**

Returns an YMeasure object which summarizes the whole DataSet.

#### **dataset→get\_unit()**

Returns the measuring unit for the measured value.



**dataset→loadMore()**

Loads the the next block of measures from the dataLogger, and updates the progress indicator.

**dataset→loadMore\_async(callback, context)**

Loads the the next block of measures from the dataLogger asynchronously.

**dataset**→**get\_endTimeUTC()****YDataSet****dataset**→**endTimeUTC()****dataset.get\_endTimeUTC()**

Returns the end time of the dataset, relative to the Jan 1, 1970.

js	function <b>get_endTimeUTC</b> ( )
nodejs	function <b>get_endTimeUTC</b> ( )
php	function <b>get_endTimeUTC</b> ( )
cpp	s64 <b>get_endTimeUTC</b> ( )
m	-(s64) endTimeUTC
pas	function <b>get_endTimeUTC</b> ( ): int64
vb	function <b>get_endTimeUTC</b> ( ) As Long
cs	long <b>get_endTimeUTC</b> ( )
java	long <b>get_endTimeUTC</b> ( )
py	def <b>get_endTimeUTC</b> ( )

When the YDataSet is created, the end time is the value passed in parameter to the `get_dataSet( )` function. After the very first call to `loadMore( )`, the end time is updated to reflect the timestamp of the last measure actually found in the `dataLogger` within the specified range.

**Returns :**

an unsigned number corresponding to the number of seconds between the Jan 1, 1970 and the end of this data set (i.e. Unix time representation of the absolute time).

**dataset**→**get\_functionId()****YDataSet****dataset**→**functionId()****dataset.get\_functionId()**

Returns the hardware identifier of the function that performed the measure, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
pas	function <b>get_functionId</b> ( ): string
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `temperature1`.

**Returns :**

a string that identifies the function (ex: `temperature1`)

**dataset**→**get\_hardwareId()****YDataSet****dataset**→**hardwareId()****dataset.get\_hardwareId()**

Returns the unique hardware identifier of the function who performed the measures, in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
pas	function <b>get_hardwareId</b> ( ): string
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the function (for example `THRMCPL1-123456.temperature1`)

**Returns :**

a string that uniquely identifies the function (ex: `THRMCPL1-123456.temperature1`)

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

**dataset**→**get\_measures()****YDataSet****dataset**→**measures()****dataset.get\_measures()**

Returns all measured values currently available for this DataSet, as a list of YMeasure objects.

js	function <b>get_measures</b> ( )
nodejs	function <b>get_measures</b> ( )
php	function <b>get_measures</b> ( )
cpp	vector<YMeasure> <b>get_measures</b> ( )
m	-(NSMutableArray*) measures
pas	function <b>get_measures</b> ( ): TYMeasureArray
vb	function <b>get_measures</b> ( ) As List
cs	List<YMeasure> <b>get_measures</b> ( )
java	ArrayList<YMeasure> <b>get_measures</b> ( )
py	def <b>get_measures</b> ( )

Each item includes: - the start of the measure time interval - the end of the measure time interval - the minimal value observed during the time interval - the average value observed during the time interval - the maximal value observed during the time interval

Before calling this method, you should call `loadMore()` to load data from the device. You may have to call `loadMore()` several time until all rows are loaded, but you can start looking at available data rows before the load is complete.

The oldest measures are always loaded first, and the most recent measures will be loaded last. As a result, timestamps are normally sorted in ascending order within the measure table, unless there was an unexpected adjustment of the datalogger UTC clock.

**Returns :**

a table of records, where each record depicts the measured value for a given time interval

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

**dataset**→**get\_preview()****YDataSet****dataset**→**preview()****dataset.get\_preview()**

Returns a condensed version of the measures that can retrieved in this YDataSet, as a list of YMeasure objects.

js	function <b>get_preview</b> ( )
nodejs	function <b>get_preview</b> ( )
php	function <b>get_preview</b> ( )
cpp	vector<YMeasure> <b>get_preview</b> ( )
m	-(NSMutableArray*) preview
pas	function <b>get_preview</b> ( ): TYMeasureArray
vb	function <b>get_preview</b> ( ) As List
cs	List<YMeasure> <b>get_preview</b> ( )
java	ArrayList<YMeasure> <b>get_preview</b> ( )
py	def <b>get_preview</b> ( )

Each item includes: - the start of a time interval - the end of a time interval - the minimal value observed during the time interval - the average value observed during the time interval - the maximal value observed during the time interval

This preview is available as soon as `loadMore( )` has been called for the first time.

**Returns :**

a table of records, where each record depicts the measured values during a time interval

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

**dataset**→**get\_progress()****YDataSet****dataset**→**progress()****dataset.get\_progress()**

Returns the progress of the downloads of the measures from the data logger, on a scale from 0 to 100.

js	function <b>get_progress</b> ( )
nodejs	function <b>get_progress</b> ( )
php	function <b>get_progress</b> ( )
cpp	int <b>get_progress</b> ( )
m	-(int) progress
pas	function <b>get_progress</b> ( ): LongInt
vb	function <b>get_progress</b> ( ) As Integer
cs	int <b>get_progress</b> ( )
java	int <b>get_progress</b> ( )
py	def <b>get_progress</b> ( )

When the object is instantiated by `get_dataSet`, the progress is zero. Each time `loadMore( )` is invoked, the progress is updated, to reach the value 100 only once all measures have been loaded.

**Returns :**

an integer in the range 0 to 100 (percentage of completion).

**dataset**→**get\_startTimeUTC()****YDataSet****dataset**→**startTimeUTC()****dataset.get\_startTimeUTC()**

Returns the start time of the dataset, relative to the Jan 1, 1970.

js	function <b>get_startTimeUTC</b> ( )
nodejs	function <b>get_startTimeUTC</b> ( )
php	function <b>get_startTimeUTC</b> ( )
cpp	s64 <b>get_startTimeUTC</b> ( )
m	-(s64) <b>startTimeUTC</b>
pas	function <b>get_startTimeUTC</b> ( ): int64
vb	function <b>get_startTimeUTC</b> ( ) As Long
cs	long <b>get_startTimeUTC</b> ( )
java	long <b>get_startTimeUTC</b> ( )
py	def <b>get_startTimeUTC</b> ( )

When the YDataSet is created, the start time is the value passed in parameter to the `get_dataSet( )` function. After the very first call to `loadMore( )`, the start time is updated to reflect the timestamp of the first measure actually found in the dataLogger within the specified range.

**Returns :**

an unsigned number corresponding to the number of seconds between the Jan 1, 1970 and the beginning of this data set (i.e. Unix time representation of the absolute time).



**dataset**→**get\_summary()****YDataSet****dataset**→**summary()****dataset.get\_summary()**

Returns an YMeasure object which summarizes the whole DataSet.

js	function <b>get_summary</b> ( )
nodejs	function <b>get_summary</b> ( )
php	function <b>get_summary</b> ( )
cpp	YMeasure <b>get_summary</b> ( )
m	-(YMeasure*) summary
pas	function <b>get_summary</b> ( ): TYMeasure
vb	function <b>get_summary</b> ( ) As YMeasure
cs	YMeasure <b>get_summary</b> ( )
java	YMeasure <b>get_summary</b> ( )
py	def <b>get_summary</b> ( )

It includes the following information: - the start of a time interval - the end of a time interval - the minimal value observed during the time interval - the average value observed during the time interval - the maximal value observed during the time interval

This summary is available as soon as `loadMore( )` has been called for the first time.

**Returns :**

an YMeasure object

**dataset**→**get\_unit()****YDataSet****dataset**→**unit()****dataset.get\_unit()**

Returns the measuring unit for the measured value.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )

**Returns :**

a string that represents a physical unit.

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**dataset→loadMore()dataset.loadMore()****YDataSet**

Loads the the next block of measures from the dataLogger, and updates the progress indicator.

js	function <b>loadMore</b> ( )
nodejs	function <b>loadMore</b> ( )
php	function <b>loadMore</b> ( )
cpp	int <b>loadMore</b> ( )
m	-(int) <b>loadMore</b>
pas	function <b>loadMore</b> ( ): LongInt
vb	function <b>loadMore</b> ( ) As Integer
cs	int <b>loadMore</b> ( )
java	int <b>loadMore</b> ( )
py	def <b>loadMore</b> ( )

**Returns :**

an integer in the range 0 to 100 (percentage of completion), or a negative error code in case of failure.

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

**dataset→loadMore\_async()****YDataSet**

Loads the the next block of measures from the dataLogger asynchronously.

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

**Parameters :**

- callback** callback function that is invoked when the w The callback function receives three arguments: - the user-specific context object - the YDataSet object whose loadMore\_async was invoked - the load result: either the progress indicator (0...100), or a negative error code in case of failure.
- context** user-specific object that is passed as-is to the callback function

**Returns :**

nothing.

## 3.11. Unformatted data sequence

YDataStream objects represent bare recorded measure sequences, exactly as found within the data logger present on Yoctopuce sensors.

In most cases, it is not necessary to use YDataStream objects directly, as the YDataSet objects (returned by the `get_recordedData()` method from sensors and the `get_dataSets()` method from the data logger) provide a more convenient interface.

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

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

YDataStream methods
<b>datastream→get_averageValue()</b> Returns the average of all measures observed within this stream.
<b>datastream→get_columnCount()</b> Returns the number of data columns present in this stream.
<b>datastream→get_columnNames()</b> Returns the title (or meaning) of each data column present in this stream.
<b>datastream→get_data(row, col)</b> Returns a single measure from the data stream, specified by its row and column index.
<b>datastream→get_dataRows()</b> Returns the whole data set contained in the stream, as a bidimensional table of numbers.
<b>datastream→get_dataSamplesIntervalMs()</b> Returns the number of milliseconds between two consecutive rows of this data stream.
<b>datastream→get_duration()</b> Returns the approximate duration of this stream, in seconds.
<b>datastream→get_maxValue()</b> Returns the largest measure observed within this stream.
<b>datastream→get_minValue()</b> Returns the smallest measure observed within this stream.
<b>datastream→get_rowCount()</b> Returns the number of data rows present in this stream.
<b>datastream→get_runIndex()</b> Returns the run index of the data stream.
<b>datastream→get_startTime()</b> Returns the relative start time of the data stream, measured in seconds.
<b>datastream→get_startTimeUTC()</b>

### 3. Reference

---

Returns the start time of the data stream, relative to the Jan 1, 1970.

**datastream→get\_averageValue()****YDataStream****datastream→averageValue()****datastream.get\_averageValue()**

Returns the average of all measures observed within this stream.

<code>js</code>	<code>function get_averageValue( )</code>
<code>nodejs</code>	<code>function get_averageValue( )</code>
<code>php</code>	<code>function get_averageValue( )</code>
<code>cpp</code>	<code>double get_averageValue( )</code>
<code>m</code>	<code>-(double) averageValue</code>
<code>pas</code>	<code>function get_averageValue( ): double</code>
<code>vb</code>	<code>function get_averageValue( ) As Double</code>
<code>cs</code>	<code>double get_averageValue( )</code>
<code>java</code>	<code>double get_averageValue( )</code>
<code>py</code>	<code>def get_averageValue( )</code>

If the device uses a firmware older than version 13000, this method will always return Y\_DATA\_INVALID.

**Returns :**

a floating-point number corresponding to the average value, or Y\_DATA\_INVALID if the stream is not yet complete (still recording).

On failure, throws an exception or returns Y\_DATA\_INVALID.

**datastream**→**get\_columnCount()****YDataStream****datastream**→**columnCount()****datastream.get\_columnCount()**

Returns the number of data columns present in this stream.

js	function <b>get_columnCount</b> ( )
nodejs	function <b>get_columnCount</b> ( )
php	function <b>get_columnCount</b> ( )
cpp	int <b>get_columnCount</b> ( )
m	-(int) columnCount
pas	function <b>get_columnCount</b> ( ): LongInt
vb	function <b>get_columnCount</b> ( ) As Integer
cs	int <b>get_columnCount</b> ( )
java	int <b>get_columnCount</b> ( )
py	def <b>get_columnCount</b> ( )

The meaning of the values present in each column can be obtained using the method `get_columnNames()`.

If the device uses a firmware older than version 13000, this method fetches the whole data stream from the device if not yet done, which can cause a little delay.

**Returns :**

an unsigned number corresponding to the number of columns.

On failure, throws an exception or returns zero.



**datastream→get\_columnNames()****YDataStream****datastream→columnNames()****datastream.get\_columnNames()**

Returns the title (or meaning) of each data column present in this stream.

js	function <b>get_columnNames</b> ( )
nodejs	function <b>get_columnNames</b> ( )
php	function <b>get_columnNames</b> ( )
cpp	vector<string> <b>get_columnNames</b> ( )
m	-(NSMutableArray*) columnNames
pas	function <b>get_columnNames</b> ( ): TStringArray
vb	function <b>get_columnNames</b> ( ) As List
cs	List<string> <b>get_columnNames</b> ( )
java	ArrayList<String> <b>get_columnNames</b> ( )
py	def <b>get_columnNames</b> ( )

In most case, the title of the data column is the hardware identifier of the sensor that produced the data. For streams recorded at a lower recording rate, the dataLogger stores the min, average and max value during each measure interval into three columns with suffixes `_min`, `_avg` and `_max` respectively.

If the device uses a firmware older than version 13000, this method fetches the whole data stream from the device if not yet done, which can cause a little delay.

**Returns :**

a list containing as many strings as there are columns in the data stream.

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

**datastream**→**get\_data()****YDataStream****datastream**→**data()****datastream.get\_data()**

Returns a single measure from the data stream, specified by its row and column index.

js	function <b>get_data</b> ( <b>row</b> , <b>col</b> )
nodejs	function <b>get_data</b> ( <b>row</b> , <b>col</b> )
php	function <b>get_data</b> ( <b>\$row</b> , <b>\$col</b> )
cpp	double <b>get_data</b> ( int <b>row</b> , int <b>col</b> )
m	-(double) data : (int) <b>row</b> : (int) <b>col</b>
pas	function <b>get_data</b> ( <b>row</b> : LongInt, <b>col</b> : LongInt): double
vb	function <b>get_data</b> ( ) As Double
cs	double <b>get_data</b> ( int <b>row</b> , int <b>col</b> )
java	double <b>get_data</b> ( int <b>row</b> , int <b>col</b> )
py	def <b>get_data</b> ( <b>row</b> , <b>col</b> )

The meaning of the values present in each column can be obtained using the method `get_columnNames()`.

This method fetches the whole data stream from the device, if not yet done.

**Parameters :**

**row** row index  
**col** column index

**Returns :**

a floating-point number

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

**datastream→get\_dataRows()****YDataStream****datastream→dataRows()datastream.get\_dataRows()**

Returns the whole data set contained in the stream, as a bidimensional table of numbers.

js	function <b>get_dataRows</b> ( )
nodejs	function <b>get_dataRows</b> ( )
php	function <b>get_dataRows</b> ( )
cpp	vector< vector<double> > <b>get_dataRows</b> ( )
m	-(NSMutableArray*) dataRows
pas	function <b>get_dataRows</b> ( ): TDoubleArrayArray
vb	function <b>get_dataRows</b> ( ) As List
cs	List<List<double>> <b>get_dataRows</b> ( )
java	ArrayList<ArrayList<Double>> <b>get_dataRows</b> ( )
py	def <b>get_dataRows</b> ( )

The meaning of the values present in each column can be obtained using the method `get_columnNames()`.

This method fetches the whole data stream from the device, if not yet done.

**Returns :**

a list containing as many elements as there are rows in the data stream. Each row itself is a list of floating-point numbers.

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

**datastream→get\_dataSamplesIntervalMs()****YDataStream****datastream→dataSamplesIntervalMs()****datastream.get\_dataSamplesIntervalMs()**

Returns the number of milliseconds between two consecutive rows of this data stream.

js	function <b>get_dataSamplesIntervalMs</b> ( )
nodejs	function <b>get_dataSamplesIntervalMs</b> ( )
php	function <b>get_dataSamplesIntervalMs</b> ( )
cpp	int <b>get_dataSamplesIntervalMs</b> ( )
m	-(int) dataSamplesIntervalMs
pas	function <b>get_dataSamplesIntervalMs</b> ( ): LongInt
vb	function <b>get_dataSamplesIntervalMs</b> ( ) As Integer
cs	int <b>get_dataSamplesIntervalMs</b> ( )
java	int <b>get_dataSamplesIntervalMs</b> ( )
py	def <b>get_dataSamplesIntervalMs</b> ( )

By default, the data logger records one row per second, but the recording frequency can be changed for each device function

**Returns :**

an unsigned number corresponding to a number of milliseconds.

**datastream**→**get\_duration()****YDataStream****datastream**→**duration()****datastream.get\_duration()**

Returns the approximate duration of this stream, in seconds.

js	function <b>get_duration</b> ( )
nodejs	function <b>get_duration</b> ( )
php	function <b>get_duration</b> ( )
cpp	int <b>get_duration</b> ( )
m	-(int) duration
pas	function <b>get_duration</b> ( ): LongInt
vb	function <b>get_duration</b> ( ) As Integer
cs	int <b>get_duration</b> ( )
java	int <b>get_duration</b> ( )
py	def <b>get_duration</b> ( )

**Returns :**

the number of seconds covered by this stream.

On failure, throws an exception or returns Y\_DURATION\_INVALID.

**datastream**→**get\_maxValue()****YDataStream****datastream**→**maxValue()****datastream.get\_maxValue()**

Returns the largest measure observed within this stream.

js	function <b>get_maxValue</b> ( )
nodejs	function <b>get_maxValue</b> ( )
php	function <b>get_maxValue</b> ( )
cpp	double <b>get_maxValue</b> ( )
m	-(double) <b>maxValue</b>
pas	function <b>get_maxValue</b> ( ): double
vb	function <b>get_maxValue</b> ( ) As Double
cs	double <b>get_maxValue</b> ( )
java	double <b>get_maxValue</b> ( )
py	def <b>get_maxValue</b> ( )

If the device uses a firmware older than version 13000, this method will always return Y\_DATA\_INVALID.

**Returns :**

a floating-point number corresponding to the largest value, or Y\_DATA\_INVALID if the stream is not yet complete (still recording).

On failure, throws an exception or returns Y\_DATA\_INVALID.

**datastream**→**get\_minValue()****YDataStream****datastream**→**minValue()****datastream.get\_minValue()**

Returns the smallest measure observed within this stream.

js	function <b>get_minValue</b> ( )
nodejs	function <b>get_minValue</b> ( )
php	function <b>get_minValue</b> ( )
cpp	double <b>get_minValue</b> ( )
m	-(double) minValue
pas	function <b>get_minValue</b> ( ): double
vb	function <b>get_minValue</b> ( ) As Double
cs	double <b>get_minValue</b> ( )
java	double <b>get_minValue</b> ( )
py	def <b>get_minValue</b> ( )

If the device uses a firmware older than version 13000, this method will always return Y\_DATA\_INVALID.

**Returns :**

a floating-point number corresponding to the smallest value, or Y\_DATA\_INVALID if the stream is not yet complete (still recording).

On failure, throws an exception or returns Y\_DATA\_INVALID.

**datastream**→**get\_rowCount()****YDataStream****datastream**→**rowCount()****datastream.getRowCount()**

Returns the number of data rows present in this stream.

js	function <b>get_rowCount</b> ( )
nodejs	function <b>get_rowCount</b> ( )
php	function <b>get_rowCount</b> ( )
cpp	int <b>get_rowCount</b> ( )
m	-(int) rowCount
pas	function <b>get_rowCount</b> ( ): LongInt
vb	function <b>get_rowCount</b> ( ) As Integer
cs	int <b>get_rowCount</b> ( )
java	int <b>get_rowCount</b> ( )
py	def <b>get_rowCount</b> ( )

If the device uses a firmware older than version 13000, this method fetches the whole data stream from the device if not yet done, which can cause a little delay.

**Returns :**

an unsigned number corresponding to the number of rows.

On failure, throws an exception or returns zero.



**datastream**→**get\_runIndex()****YDataStream****datastream**→**runIndex()****datastream.get\_runIndex()**

Returns the run index of the data stream.

js	function <b>get_runIndex</b> ( )
nodejs	function <b>get_runIndex</b> ( )
php	function <b>get_runIndex</b> ( )
cpp	int <b>get_runIndex</b> ( )
m	-(int) runIndex
pas	function <b>get_runIndex</b> ( ): LongInt
vb	function <b>get_runIndex</b> ( ) As Integer
cs	int <b>get_runIndex</b> ( )
java	int <b>get_runIndex</b> ( )
py	def <b>get_runIndex</b> ( )

A run can be made of multiple datastreams, for different time intervals.

**Returns :**

an unsigned number corresponding to the run index.

**datastream**→**get\_startTime()****YDataStream****datastream**→**startTime()****datastream.get\_startTime()**

Returns the relative start time of the data stream, measured in seconds.

js	function <b>get_startTime</b> ( )
nodejs	function <b>get_startTime</b> ( )
php	function <b>get_startTime</b> ( )
cpp	int <b>get_startTime</b> ( )
m	-(int) startTime
pas	function <b>get_startTime</b> ( ): LongInt
vb	function <b>get_startTime</b> ( ) As Integer
cs	int <b>get_startTime</b> ( )
java	int <b>get_startTime</b> ( )
py	def <b>get_startTime</b> ( )

For recent firmwares, the value is relative to the present time, which means the value is always negative. If the device uses a firmware older than version 13000, value is relative to the start of the time the device was powered on, and is always positive. If you need an absolute UTC timestamp, use `get_startTimeUTC()`.

**Returns :**

an unsigned number corresponding to the number of seconds between the start of the run and the beginning of this data stream.

**datastream→get\_startTimeUTC()****YDataStream****datastream→startTimeUTC()****datastream.get\_startTimeUTC()**

Returns the start time of the data stream, relative to the Jan 1, 1970.

<code>js</code>	<code>function get_startTimeUTC( )</code>
<code>nodejs</code>	<code>function get_startTimeUTC( )</code>
<code>php</code>	<code>function get_startTimeUTC( )</code>
<code>cpp</code>	<code>s64 get_startTimeUTC( )</code>
<code>m</code>	<code>-(s64) startTimeUTC</code>
<code>pas</code>	<code>function get_startTimeUTC( ): int64</code>
<code>vb</code>	<code>function get_startTimeUTC( ) As Long</code>
<code>cs</code>	<code>long get_startTimeUTC( )</code>
<code>java</code>	<code>long get_startTimeUTC( )</code>
<code>py</code>	<code>def get_startTimeUTC( )</code>

If the UTC time was not set in the datalogger at the time of the recording of this data stream, this method returns 0.

**Returns :**

an unsigned number corresponding to the number of seconds between the Jan 1, 1970 and the beginning of this data stream (i.e. Unix time representation of the absolute time).

## 3.12. Digital IO function interface

The Yoctopuce application programming interface allows you to switch the state of each bit of the I/O port. You can switch all bits at once, or one by one. The library can also automatically generate short pulses of a determined duration. Electrical behavior of each I/O can be modified (open drain and reverse polarity).

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

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

### Global functions

#### yFindDigitalIO(func)

Retrieves a digital IO port for a given identifier.

#### yFirstDigitalIO()

Starts the enumeration of digital IO ports currently accessible.

### YDigitalIO methods

#### digitalio→delayedPulse(bitno, ms\_delay, ms\_duration)

Schedules a pulse on a single bit for a specified duration.

#### digitalio→describe()

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

#### digitalio→get\_advertisedValue()

Returns the current value of the digital IO port (no more than 6 characters).

#### digitalio→get\_bitDirection(bitno)

Returns the direction of a single bit from the I/O port (0 means the bit is an input, 1 an output).

#### digitalio→get\_bitOpenDrain(bitno)

Returns the type of electrical interface of a single bit from the I/O port.

#### digitalio→get\_bitPolarity(bitno)

Returns the polarity of a single bit from the I/O port (0 means the I/O works in regular mode, 1 means the I/O works in reverse mode).

#### digitalio→get\_bitState(bitno)

Returns the state of a single bit of the I/O port.

#### digitalio→get\_errorMessage()

Returns the error message of the latest error with the digital IO port.

#### digitalio→get\_errorType()

Returns the numerical error code of the latest error with the digital IO port.

#### digitalio→get\_friendlyName()

Returns a global identifier of the digital IO port in the format MODULE\_NAME . FUNCTION\_NAME.

**digitalio→get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

**digitalio→get\_functionId()**

Returns the hardware identifier of the digital IO port, without reference to the module.

**digitalio→get\_hardwareId()**

Returns the unique hardware identifier of the digital IO port in the form SERIAL.FUNCTIONID.

**digitalio→get\_logicalName()**

Returns the logical name of the digital IO port.

**digitalio→get\_module()**

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

**digitalio→get\_module\_async(callback, context)**

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

**digitalio→get\_outputVoltage()**

Returns the voltage source used to drive output bits.

**digitalio→get\_portDirection()**

Returns the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output.

**digitalio→get\_portOpenDrain()**

Returns the electrical interface for each bit of the port.

**digitalio→get\_portPolarity()**

Returns the polarity of all the bits of the port.

**digitalio→get\_portSize()**

Returns the number of bits implemented in the I/O port.

**digitalio→get\_portState()**

Returns the digital IO port state: bit 0 represents input 0, and so on.

**digitalio→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**digitalio→isOnline()**

Checks if the digital IO port is currently reachable, without raising any error.

**digitalio→isOnline\_async(callback, context)**

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

**digitalio→load(msValidity)**

Preloads the digital IO port cache with a specified validity duration.

**digitalio→load\_async(msValidity, callback, context)**

Preloads the digital IO port cache with a specified validity duration (asynchronous version).

**digitalio→nextDigitalIO()**

Continues the enumeration of digital IO ports started using yFirstDigitalIO().

**digitalio→pulse(bitno, ms\_duration)**

Triggers a pulse on a single bit for a specified duration.

**digitalio→registerValueCallback(callback)**

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

**digitalio→set\_bitDirection(bitno, bitdirection)**

Changes the direction of a single bit from the I/O port.

**digitalio→set\_bitOpenDrain(bitno, opendrain)**

Changes the electrical interface of a single bit from the I/O port.

**digitalio→set\_bitPolarity(bitno, bitpolarity)**

### 3. Reference

Changes the polarity of a single bit from the I/O port.

**digitalio**→**set\_bitState**(**bitno**, **bitstate**)

Sets a single bit of the I/O port.

**digitalio**→**set\_logicalName**(**newval**)

Changes the logical name of the digital IO port.

**digitalio**→**set\_outputVoltage**(**newval**)

Changes the voltage source used to drive output bits.

**digitalio**→**set\_portDirection**(**newval**)

Changes the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output.

**digitalio**→**set\_portOpenDrain**(**newval**)

Changes the electrical interface for each bit of the port.

**digitalio**→**set\_portPolarity**(**newval**)

Changes the polarity of all the bits of the port: 0 makes a bit an input, 1 makes it an output.

**digitalio**→**set\_portState**(**newval**)

Changes the digital IO port state: bit 0 represents input 0, and so on.

**digitalio**→**set\_userData**(**data**)

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

**digitalio**→**toggle\_bitState**(**bitno**)

Reverts a single bit of the I/O port.

**digitalio**→**wait\_async**(**callback**, **context**)

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

## YDigitalIO.FindDigitalIO() yFindDigitalIO()YDigitalIO.FindDigitalIO()

## YDigitalIO

Retrieves a digital IO port for a given identifier.

js	function <b>yFindDigitalIO</b> ( <b>func</b> )
nodejs	function <b>FindDigitalIO</b> ( <b>func</b> )
php	function <b>yFindDigitalIO</b> ( <b>\$func</b> )
cpp	YDigitalIO* <b>yFindDigitalIO</b> ( const string& <b>func</b> )
m	YDigitalIO* <b>yFindDigitalIO</b> ( NSString* <b>func</b> )
pas	function <b>yFindDigitalIO</b> ( <b>func</b> : string): TYDigitalIO
vb	function <b>yFindDigitalIO</b> ( ByVal <b>func</b> As String) As YDigitalIO
cs	YDigitalIO <b>FindDigitalIO</b> ( string <b>func</b> )
java	YDigitalIO <b>FindDigitalIO</b> ( String <b>func</b> )
py	def <b>FindDigitalIO</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the digital IO port is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YDigitalIO.isOnline()` to test if the digital IO port is indeed online at a given time. In case of ambiguity when looking for a digital IO 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 digital IO port

### Returns :

a `YDigitalIO` object allowing you to drive the digital IO port.

## YDigitalIO.FirstDigitalIO() yFirstDigitalIO()YDigitalIO.FirstDigitalIO()

YDigitalIO

Starts the enumeration of digital IO ports currently accessible.

js	function <b>yFirstDigitalIO</b> ( )
nodejs	function <b>FirstDigitalIO</b> ( )
php	function <b>yFirstDigitalIO</b> ( )
cpp	YDigitalIO* <b>yFirstDigitalIO</b> ( )
m	YDigitalIO* <b>yFirstDigitalIO</b> ( )
pas	function <b>yFirstDigitalIO</b> ( ): TYDigitalIO
vb	function <b>yFirstDigitalIO</b> ( ) As YDigitalIO
cs	YDigitalIO <b>FirstDigitalIO</b> ( )
java	YDigitalIO <b>FirstDigitalIO</b> ( )
py	def <b>FirstDigitalIO</b> ( )

Use the method `YDigitalIO.nextDigitalIO( )` to iterate on next digital IO ports.

### Returns :

a pointer to a `YDigitalIO` object, corresponding to the first digital IO port currently online, or a `null` pointer if there are none.



**digitalio→delayedPulse()digitalio.delayedPulse()****YDigitalIO**

Schedules a pulse on a single bit for a specified duration.

js	function <b>delayedPulse</b> ( <b>bitno</b> , <b>ms_delay</b> , <b>ms_duration</b> )
nodejs	function <b>delayedPulse</b> ( <b>bitno</b> , <b>ms_delay</b> , <b>ms_duration</b> )
php	function <b>delayedPulse</b> ( <b>\$bitno</b> , <b>\$ms_delay</b> , <b>\$ms_duration</b> )
cpp	int <b>delayedPulse</b> ( int <b>bitno</b> , int <b>ms_delay</b> , int <b>ms_duration</b> )
m	-(int) <b>delayedPulse</b> : (int) <b>bitno</b> : (int) <b>ms_delay</b> : (int) <b>ms_duration</b>
pas	function <b>delayedPulse</b> ( <b>bitno</b> : LongInt, <b>ms_delay</b> : LongInt, <b>ms_duration</b> : LongInt): LongInt
vb	function <b>delayedPulse</b> ( ) As Integer
cs	int <b>delayedPulse</b> ( int <b>bitno</b> , int <b>ms_delay</b> , int <b>ms_duration</b> )
java	int <b>delayedPulse</b> ( int <b>bitno</b> , int <b>ms_delay</b> , int <b>ms_duration</b> )
py	def <b>delayedPulse</b> ( <b>bitno</b> , <b>ms_delay</b> , <b>ms_duration</b> )
cmd	YDigitalIO <b>target</b> <b>delayedPulse</b> <b>bitno</b> <b>ms_delay</b> <b>ms_duration</b>

The specified bit will be turned to 1, and then back to 0 after the given duration.

**Parameters :**

**bitno**            the bit number; lowest bit has index 0

**ms\_delay**        waiting time before the pulse, in milliseconds

**ms\_duration**    desired pulse duration in milliseconds. Be aware that the device time resolution is not guaranteed up to the millisecond.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio→describe()digitalio.describe()****YDigitalIO**

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

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

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 digital IO port (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**digitalio**→**get\_advertisedValue()****YDigitalIO****digitalio**→**advertisedValue()****digitalio.get\_advertisedValue()**

Returns the current value of the digital IO port (no more than 6 characters).

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

**Returns :**

a string corresponding to the current value of the digital IO port (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**digitalio**→**get\_bitDirection()****YDigitalIO****digitalio**→**bitDirection()****digitalio.get\_bitDirection()**

Returns the direction of a single bit from the I/O port (0 means the bit is an input, 1 an output).

js	function <b>get_bitDirection</b> ( <b>bitno</b> )
nodejs	function <b>get_bitDirection</b> ( <b>bitno</b> )
php	function <b>get_bitDirection</b> ( <b>\$bitno</b> )
cpp	int <b>get_bitDirection</b> ( int <b>bitno</b> )
m	-(int) bitDirection : (int) <b>bitno</b>
pas	function <b>get_bitDirection</b> ( <b>bitno</b> : LongInt): LongInt
vb	function <b>get_bitDirection</b> ( ) As Integer
cs	int <b>get_bitDirection</b> ( int <b>bitno</b> )
java	int <b>get_bitDirection</b> ( int <b>bitno</b> )
py	def <b>get_bitDirection</b> ( <b>bitno</b> )
cmd	YDigitalIO <b>target</b> <b>get_bitDirection</b> <b>bitno</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**get\_bitOpenDrain()****YDigitalIO****digitalio**→**bitOpenDrain()****digitalio.get\_bitOpenDrain()**

Returns the type of electrical interface of a single bit from the I/O port.

js	function <b>get_bitOpenDrain</b> ( <b>bitno</b> )
nodejs	function <b>get_bitOpenDrain</b> ( <b>bitno</b> )
php	function <b>get_bitOpenDrain</b> ( <b>\$bitno</b> )
cpp	int <b>get_bitOpenDrain</b> ( int <b>bitno</b> )
m	-(int) <b>bitOpenDrain</b> : (int) <b>bitno</b>
pas	function <b>get_bitOpenDrain</b> ( <b>bitno</b> : LongInt): LongInt
vb	function <b>get_bitOpenDrain</b> ( ) As Integer
cs	int <b>get_bitOpenDrain</b> ( int <b>bitno</b> )
java	int <b>get_bitOpenDrain</b> ( int <b>bitno</b> )
py	def <b>get_bitOpenDrain</b> ( <b>bitno</b> )
cmd	<b>YDigitalIO target get_bitOpenDrain bitno</b>

(0 means the bit is an input, 1 an output).

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**Returns :**

0 means the a bit is a regular input/output, 1 means the bit is an open-drain (open-collector) input/output.

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

**digitalio**→**get\_bitPolarity()****YDigitalIO****digitalio**→**bitPolarity()****digitalio.get\_bitPolarity()**

Returns the polarity of a single bit from the I/O port (0 means the I/O works in regular mode, 1 means the I/O works in reverse mode).

js	function <b>get_bitPolarity</b> ( <b>bitno</b> )
nodejs	function <b>get_bitPolarity</b> ( <b>bitno</b> )
php	function <b>get_bitPolarity</b> ( <b>\$bitno</b> )
cpp	int <b>get_bitPolarity</b> ( int <b>bitno</b> )
m	-(int) bitPolarity : (int) <b>bitno</b>
pas	function <b>get_bitPolarity</b> ( <b>bitno</b> : LongInt): LongInt
vb	function <b>get_bitPolarity</b> ( ) As Integer
cs	int <b>get_bitPolarity</b> ( int <b>bitno</b> )
java	int <b>get_bitPolarity</b> ( int <b>bitno</b> )
py	def <b>get_bitPolarity</b> ( <b>bitno</b> )
cmd	YDigitalIO <b>target</b> <b>get_bitPolarity</b> <b>bitno</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**get\_bitState()****YDigitalIO****digitalio**→**bitState()****digitalio.get\_bitState()**

Returns the state of a single bit of the I/O port.

js	function <b>get_bitState</b> ( <b>bitno</b> )
nodejs	function <b>get_bitState</b> ( <b>bitno</b> )
php	function <b>get_bitState</b> ( <b>\$bitno</b> )
cpp	int <b>get_bitState</b> ( int <b>bitno</b> )
m	-(int) <b>bitState</b> : (int) <b>bitno</b>
pas	function <b>get_bitState</b> ( <b>bitno</b> : LongInt): LongInt
vb	function <b>get_bitState</b> ( ) As Integer
cs	int <b>get_bitState</b> ( int <b>bitno</b> )
java	int <b>get_bitState</b> ( int <b>bitno</b> )
py	def <b>get_bitState</b> ( <b>bitno</b> )
cmd	YDigitalIO <b>target</b> <b>get_bitState</b> <b>bitno</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**Returns :**

the bit state (0 or 1)

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

**digitalio**→**get\_errorMessage()****YDigitalIO****digitalio**→**errorMessage()****digitalio.errorMessage()**

Returns the error message of the latest error with the digital IO port.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the digital IO port object



**digitalio**→**get\_errorType()****YDigitalIO****digitalio**→**errorType()****digitalio.get\_errorType()**

Returns the numerical error code of the latest error with the digital IO port.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the digital IO port object

**digitalio**→**get\_friendlyName()****YDigitalIO****digitalio**→**friendlyName()****digitalio.get\_friendlyName()**

Returns a global identifier of the digital IO port in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the digital IO port using logical names (ex: `MyCustomName.relay1`)

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

**digitalio**→**get\_functionDescriptor()****YDigitalIO****digitalio**→**functionDescriptor()****digitalio.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**digitalio**→**get\_functionId()****YDigitalIO****digitalio**→**functionId()****digitalio.get\_functionId()**

Returns the hardware identifier of the digital IO port, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the digital IO port (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**digitalio**→**get\_hardwareId()****YDigitalIO****digitalio**→**hardwareId()****digitalio.get\_hardwareId()**

Returns the unique hardware identifier of the digital IO port in the form `SERIAL.FUNCTIONID`.

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

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

**Returns :**

a string that uniquely identifies the digital IO port (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**digitalio**→**get\_logicalName()****YDigitalIO****digitalio**→**logicalName()****digitalio.get\_logicalName()**

Returns the logical name of the digital IO port.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the digital IO port. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**digitalio**→**get\_module()****YDigitalIO****digitalio**→**module()****digitalio.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**digitalio**→**get\_module\_async()****YDigitalIO****digitalio**→**module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.



**digitalio→get\_outputVoltage()****YDigitalIO****digitalio→outputVoltage()****digitalio.get\_outputVoltage()**

Returns the voltage source used to drive output bits.

js	function <b>get_outputVoltage</b> ( )
nodejs	function <b>get_outputVoltage</b> ( )
php	function <b>get_outputVoltage</b> ( )
cpp	Y_OUTPUTVOLTAGE_enum <b>get_outputVoltage</b> ( )
m	-(Y_OUTPUTVOLTAGE_enum) outputVoltage
pas	function <b>get_outputVoltage</b> ( ): Integer
vb	function <b>get_outputVoltage</b> ( ) As Integer
cs	int <b>get_outputVoltage</b> ( )
java	int <b>get_outputVoltage</b> ( )
py	def <b>get_outputVoltage</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_outputVoltage</b>

**Returns :**

a value among Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V and Y\_OUTPUTVOLTAGE\_EXT\_V corresponding to the voltage source used to drive output bits

On failure, throws an exception or returns Y\_OUTPUTVOLTAGE\_INVALID.

**digitalio**→**get\_portDirection()****YDigitalIO****digitalio**→**portDirection()****digitalio.get\_portDirection()**

Returns the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output.

js	function <b>get_portDirection</b> ( )
nodejs	function <b>get_portDirection</b> ( )
php	function <b>get_portDirection</b> ( )
cpp	int <b>get_portDirection</b> ( )
m	-(int) portDirection
pas	function <b>get_portDirection</b> ( ): LongInt
vb	function <b>get_portDirection</b> ( ) As Integer
cs	int <b>get_portDirection</b> ( )
java	int <b>get_portDirection</b> ( )
py	def <b>get_portDirection</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_portDirection</b>

**Returns :**

an integer corresponding to the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output

On failure, throws an exception or returns Y\_PORTDIRECTION\_INVALID.

**digitalio→get\_portOpenDrain()****YDigitalIO****digitalio→portOpenDrain()****digitalio.get\_portOpenDrain()**

Returns the electrical interface for each bit of the port.

js	function <b>get_portOpenDrain</b> ( )
nodejs	function <b>get_portOpenDrain</b> ( )
php	function <b>get_portOpenDrain</b> ( )
cpp	int <b>get_portOpenDrain</b> ( )
m	-(int) portOpenDrain
pas	function <b>get_portOpenDrain</b> ( ): LongInt
vb	function <b>get_portOpenDrain</b> ( ) As Integer
cs	int <b>get_portOpenDrain</b> ( )
java	int <b>get_portOpenDrain</b> ( )
py	def <b>get_portOpenDrain</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_portOpenDrain</b>

For each bit set to 0 the matching I/O works in the regular, intuitive way, for each bit set to 1, the I/O works in reverse mode.

**Returns :**

an integer corresponding to the electrical interface for each bit of the port

On failure, throws an exception or returns Y\_PORTOPENDRAIN\_INVALID.

**digitalio**→**get\_portPolarity()****YDigitalIO****digitalio**→**portPolarity()****digitalio.get\_portPolarity()**

Returns the polarity of all the bits of the port.

js	function <b>get_portPolarity</b> ( )
nodejs	function <b>get_portPolarity</b> ( )
php	function <b>get_portPolarity</b> ( )
cpp	int <b>get_portPolarity</b> ( )
m	-(int) portPolarity
pas	function <b>get_portPolarity</b> ( ): LongInt
vb	function <b>get_portPolarity</b> ( ) As Integer
cs	int <b>get_portPolarity</b> ( )
java	int <b>get_portPolarity</b> ( )
py	def <b>get_portPolarity</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_portPolarity</b>

For each bit set to 0, the matching I/O works the regular, intuitive way; for each bit set to 1, the I/O works in reverse mode.

**Returns :**

an integer corresponding to the polarity of all the bits of the port

On failure, throws an exception or returns Y\_PORTPOLARITY\_INVALID.

**digitalio**→**get\_portSize()****YDigitalIO****digitalio**→**portSize()****digitalio.get\_portSize()**

Returns the number of bits implemented in the I/O port.

js	function <b>get_portSize</b> ( )
nodejs	function <b>get_portSize</b> ( )
php	function <b>get_portSize</b> ( )
cpp	int <b>get_portSize</b> ( )
m	-(int) portSize
pas	function <b>get_portSize</b> ( ): LongInt
vb	function <b>get_portSize</b> ( ) As Integer
cs	int <b>get_portSize</b> ( )
java	int <b>get_portSize</b> ( )
py	def <b>get_portSize</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_portSize</b>

**Returns :**

an integer corresponding to the number of bits implemented in the I/O port

On failure, throws an exception or returns Y\_PORTSIZE\_INVALID.

**digitalio**→**get\_portState()****YDigitalIO****digitalio**→**portState()****digitalio.get\_portState()**

Returns the digital IO port state: bit 0 represents input 0, and so on.

js	function <b>get_portState</b> ( )
nodejs	function <b>get_portState</b> ( )
php	function <b>get_portState</b> ( )
cpp	int <b>get_portState</b> ( )
m	-(int) portState
pas	function <b>get_portState</b> ( ): LongInt
vb	function <b>get_portState</b> ( ) As Integer
cs	int <b>get_portState</b> ( )
java	int <b>get_portState</b> ( )
py	def <b>get_portState</b> ( )
cmd	YDigitalIO <b>target</b> <b>get_portState</b>

**Returns :**

an integer corresponding to the digital IO port state: bit 0 represents input 0, and so on

On failure, throws an exception or returns Y\_PORTSTATE\_INVALID.

**digitalio**→**get\_userData()****YDigitalIO****digitalio**→**userData()****digitalio.get\_userData()**

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

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

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.

Checks if the digital IO port is currently reachable, without raising any error.

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

If there is a cached value for the digital IO 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 digital IO port.

**Returns :**

`true` if the digital IO port can be reached, and `false` otherwise



**digitalio→isOnline\_async()****YDigitalIO**

Checks if the digital IO 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 digital IO 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.

Preloads the digital IO port cache with a specified validity duration.

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

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 traffic 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.

**digitalio→load\_async()****YDigitalIO**

Preloads the digital IO 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 traffic 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.

**digitalio**→**nextDigitalIO()****digitalio.nextDigitalIO()****YDigitalIO**

Continues the enumeration of digital IO ports started using `yFirstDigitalIO()`.

js	function <b>nextDigitalIO</b> ( )
nodejs	function <b>nextDigitalIO</b> ( )
php	function <b>nextDigitalIO</b> ( )
cpp	YDigitalIO * <b>nextDigitalIO</b> ( )
m	-(YDigitalIO*) <b>nextDigitalIO</b>
pas	function <b>nextDigitalIO</b> ( ): TYDigitalIO
vb	function <b>nextDigitalIO</b> ( ) As YDigitalIO
cs	YDigitalIO <b>nextDigitalIO</b> ( )
java	YDigitalIO <b>nextDigitalIO</b> ( )
py	def <b>nextDigitalIO</b> ( )

**Returns :**

a pointer to a `YDigitalIO` object, corresponding to a digital IO port currently online, or a `null` pointer if there are no more digital IO ports to enumerate.

**digitalio→pulse()digitalio.pulse()****YDigitalIO**

Triggers a pulse on a single bit for a specified duration.

js	function <b>pulse</b> ( <b>bitno</b> , <b>ms_duration</b> )
nodejs	function <b>pulse</b> ( <b>bitno</b> , <b>ms_duration</b> )
php	function <b>pulse</b> ( <b>\$bitno</b> , <b>\$ms_duration</b> )
cpp	int <b>pulse</b> ( int <b>bitno</b> , int <b>ms_duration</b> )
m	-(int) <b>pulse</b> : (int) <b>bitno</b> : (int) <b>ms_duration</b>
pas	function <b>pulse</b> ( <b>bitno</b> : LongInt, <b>ms_duration</b> : LongInt): LongInt
vb	function <b>pulse</b> ( ) As Integer
cs	int <b>pulse</b> ( int <b>bitno</b> , int <b>ms_duration</b> )
java	int <b>pulse</b> ( int <b>bitno</b> , int <b>ms_duration</b> )
py	def <b>pulse</b> ( <b>bitno</b> , <b>ms_duration</b> )
cmd	YDigitalIO <b>target pulse bitno ms_duration</b>

The specified bit will be turned to 1, and then back to 0 after the given duration.

**Parameters :**

- bitno** the bit number; lowest bit has index 0
- ms\_duration** desired pulse duration in milliseconds. Be aware that the device time resolution is not guaranteed up to the millisecond.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## digitalio→registerValueCallback() digitalio.registerValueCallback()

YDigitalIO

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YDigitalIOValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YDigitalIOValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYDigitalIOValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**digitalio**→**set\_bitDirection()****YDigitalIO****digitalio**→**setBitDirection()****digitalio.set\_bitDirection()**

Changes the direction of a single bit from the I/O port.

js	function <b>set_bitDirection</b> ( <b>bitno</b> , <b>bitdirection</b> )
nodejs	function <b>set_bitDirection</b> ( <b>bitno</b> , <b>bitdirection</b> )
php	function <b>set_bitDirection</b> ( <b>\$bitno</b> , <b>\$bitdirection</b> )
cpp	int <b>set_bitDirection</b> ( int <b>bitno</b> , int <b>bitdirection</b> )
m	-(int) <b>setBitDirection</b> : (int) <b>bitno</b> : (int) <b>bitdirection</b>
pas	function <b>set_bitDirection</b> ( <b>bitno</b> : LongInt, <b>bitdirection</b> : LongInt): LongInt
vb	function <b>set_bitDirection</b> ( ) As Integer
cs	int <b>set_bitDirection</b> ( int <b>bitno</b> , int <b>bitdirection</b> )
java	int <b>set_bitDirection</b> ( int <b>bitno</b> , int <b>bitdirection</b> )
py	def <b>set_bitDirection</b> ( <b>bitno</b> , <b>bitdirection</b> )
cmd	YDigitalIO <b>target</b> <b>set_bitDirection</b> <b>bitno</b> <b>bitdirection</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**bitdirection** direction to set, 0 makes the bit an input, 1 makes it an output. Remember to call the `saveToFlash( )` method to make sure the setting is kept after a reboot.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**set\_bitOpenDrain()****digitalio**→**setBitOpenDrain()****digitalio.set\_bitOpenDrain()**

Changes the electrical interface of a single bit from the I/O port.

js	function <b>set_bitOpenDrain</b> ( <b>bitno</b> , <b>opendrain</b> )
nodejs	function <b>set_bitOpenDrain</b> ( <b>bitno</b> , <b>opendrain</b> )
php	function <b>set_bitOpenDrain</b> ( <b>\$bitno</b> , <b>\$opendrain</b> )
cpp	int <b>set_bitOpenDrain</b> ( int <b>bitno</b> , int <b>opendrain</b> )
m	-(int) setBitOpenDrain : (int) <b>bitno</b> : (int) <b>opendrain</b>
pas	function <b>set_bitOpenDrain</b> ( <b>bitno</b> : LongInt, <b>opendrain</b> : LongInt): LongInt
vb	function <b>set_bitOpenDrain</b> ( ) As Integer
cs	int <b>set_bitOpenDrain</b> ( int <b>bitno</b> , int <b>opendrain</b> )
java	int <b>set_bitOpenDrain</b> ( int <b>bitno</b> , int <b>opendrain</b> )
py	def <b>set_bitOpenDrain</b> ( <b>bitno</b> , <b>opendrain</b> )
cmd	YDigitalIO <b>target set_bitOpenDrain bitno opendrain</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**opendrain** 0 makes a bit a regular input/output, 1 makes it an open-drain (open-collector) input/output. Remember to call the `saveToFlash( )` method to make sure the setting is kept after a reboot.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**digitalio**→**set\_bitPolarity()****YDigitalIO****digitalio**→**setBitPolarity()****digitalio.set\_bitPolarity()**

Changes the polarity of a single bit from the I/O port.

js	function <b>set_bitPolarity</b> ( <b>bitno</b> , <b>bitpolarity</b> )
nodejs	function <b>set_bitPolarity</b> ( <b>bitno</b> , <b>bitpolarity</b> )
php	function <b>set_bitPolarity</b> ( <b>\$bitno</b> , <b>\$bitpolarity</b> )
cpp	int <b>set_bitPolarity</b> ( int <b>bitno</b> , int <b>bitpolarity</b> )
m	-(int) <b>setBitPolarity</b> : (int) <b>bitno</b> : (int) <b>bitpolarity</b>
pas	function <b>set_bitPolarity</b> ( <b>bitno</b> : LongInt, <b>bitpolarity</b> : LongInt): LongInt
vb	function <b>set_bitPolarity</b> ( ) As Integer
cs	int <b>set_bitPolarity</b> ( int <b>bitno</b> , int <b>bitpolarity</b> )
java	int <b>set_bitPolarity</b> ( int <b>bitno</b> , int <b>bitpolarity</b> )
py	def <b>set_bitPolarity</b> ( <b>bitno</b> , <b>bitpolarity</b> )
cmd	<b>YDigitalIO target set_bitPolarity bitno bitpolarity</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0.

**bitpolarity** polarity to set, 0 makes the I/O work in regular mode, 1 makes the I/O works in reverse mode.  
Remember to call the `saveToFlash( )` method to make sure the setting is kept after a reboot.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**set\_bitState()****YDigitalIO****digitalio**→**setBitState()****digitalio.set\_bitState()**

Sets a single bit of the I/O port.

js	function <b>set_bitState</b> ( <b>bitno</b> , <b>bitstate</b> )
nodejs	function <b>set_bitState</b> ( <b>bitno</b> , <b>bitstate</b> )
php	function <b>set_bitState</b> ( <b>\$bitno</b> , <b>\$bitstate</b> )
cpp	int <b>set_bitState</b> ( int <b>bitno</b> , int <b>bitstate</b> )
m	-(int) setBitState : (int) <b>bitno</b> : (int) <b>bitstate</b>
pas	function <b>set_bitState</b> ( <b>bitno</b> : LongInt, <b>bitstate</b> : LongInt): LongInt
vb	function <b>set_bitState</b> ( ) As Integer
cs	int <b>set_bitState</b> ( int <b>bitno</b> , int <b>bitstate</b> )
java	int <b>set_bitState</b> ( int <b>bitno</b> , int <b>bitstate</b> )
py	def <b>set_bitState</b> ( <b>bitno</b> , <b>bitstate</b> )
cmd	YDigitalIO <b>target set_bitState</b> bitno bitstate

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**bitstate** the state of the bit (1 or 0)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**set\_logicalName()****YDigitalIO****digitalio**→**setLogicalName()****digitalio.set\_logicalName()**

Changes the logical name of the digital IO port.

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

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 digital IO port.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**digitalio→set\_outputVoltage()**  
**digitalio→setOutputVoltage()**  
**digitalio.set\_outputVoltage()**

YDigitalIO

Changes the voltage source used to drive output bits.

js	function <b>set_outputVoltage</b> ( <b>newval</b> )
nodejs	function <b>set_outputVoltage</b> ( <b>newval</b> )
php	function <b>set_outputVoltage</b> ( <b>\$newval</b> )
cpp	int <b>set_outputVoltage</b> ( Y_OUTPUTVOLTAGE_enum <b>newval</b> )
m	-(int) setOutputVoltage : (Y_OUTPUTVOLTAGE_enum) <b>newval</b>
pas	function <b>set_outputVoltage</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_outputVoltage</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_outputVoltage</b> ( int <b>newval</b> )
java	int <b>set_outputVoltage</b> ( int <b>newval</b> )
py	def <b>set_outputVoltage</b> ( <b>newval</b> )
cmd	YDigitalIO <b>target</b> <b>set_outputVoltage</b> <b>newval</b>

Remember to call the `saveToFlash( )` method to make sure the setting is kept after a reboot.

#### Parameters :

**newval** a value among Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V and Y\_OUTPUTVOLTAGE\_EXT\_V corresponding to the voltage source used to drive output bits

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**digitalio→set\_portDirection()**  
**digitalio→setPortDirection()**  
**digitalio.set\_portDirection()**

YDigitalIO

Changes the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output.

js	function <b>set_portDirection</b> ( <b>newval</b> )
nodejs	function <b>set_portDirection</b> ( <b>newval</b> )
php	function <b>set_portDirection</b> ( <b>\$newval</b> )
cpp	int <b>set_portDirection</b> ( int <b>newval</b> )
m	-(int) setPortDirection : (int) <b>newval</b>
pas	function <b>set_portDirection</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_portDirection</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_portDirection</b> ( int <b>newval</b> )
java	int <b>set_portDirection</b> ( int <b>newval</b> )
py	def <b>set_portDirection</b> ( <b>newval</b> )
cmd	YDigitalIO <b>target</b> <b>set_portDirection</b> <b>newval</b>

Remember to call the `saveToFlash()` method to make sure the setting is kept after a reboot.

#### Parameters :

**newval** an integer corresponding to the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**digitalio→set\_portOpenDrain()****YDigitalIO****digitalio→setPortOpenDrain()****digitalio.set\_portOpenDrain()**

Changes the electrical interface for each bit of the port.

js	function <b>set_portOpenDrain</b> ( <b>newval</b> )
nodejs	function <b>set_portOpenDrain</b> ( <b>newval</b> )
php	function <b>set_portOpenDrain</b> ( <b>\$newval</b> )
cpp	int <b>set_portOpenDrain</b> ( int <b>newval</b> )
m	-(int) setPortOpenDrain : (int) <b>newval</b>
pas	function <b>set_portOpenDrain</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_portOpenDrain</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_portOpenDrain</b> ( int <b>newval</b> )
java	int <b>set_portOpenDrain</b> ( int <b>newval</b> )
py	def <b>set_portOpenDrain</b> ( <b>newval</b> )
cmd	YDigitalIO <b>target</b> <b>set_portOpenDrain</b> <b>newval</b>

0 makes a bit a regular input/output, 1 makes it an open-drain (open-collector) input/output. Remember to call the `saveToFlash()` method to make sure the setting is kept after a reboot.

**Parameters :**

**newval** an integer corresponding to the electrical interface for each bit of the port

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio→set\_portPolarity()****YDigitalIO****digitalio→setPortPolarity()digitalio.set\_portPolarity()**

Changes the polarity of all the bits of the port: 0 makes a bit an input, 1 makes it an output.

js	function <b>set_portPolarity</b> ( <b>newval</b> )
nodejs	function <b>set_portPolarity</b> ( <b>newval</b> )
php	function <b>set_portPolarity</b> ( <b>\$newval</b> )
cpp	int <b>set_portPolarity</b> ( int <b>newval</b> )
m	-(int) setPortPolarity : (int) <b>newval</b>
pas	function <b>set_portPolarity</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_portPolarity</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_portPolarity</b> ( int <b>newval</b> )
java	int <b>set_portPolarity</b> ( int <b>newval</b> )
py	def <b>set_portPolarity</b> ( <b>newval</b> )
cmd	YDigitalIO <b>target set_portPolarity newval</b>

Remember to call the `saveToFlash( )` method to make sure the setting will be kept after a reboot.

**Parameters :**

**newval** an integer corresponding to the polarity of all the bits of the port: 0 makes a bit an input, 1 makes it an output

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**set\_portState()****YDigitalIO****digitalio**→**setPortState()****digitalio.set\_portState()**

Changes the digital IO port state: bit 0 represents input 0, and so on.

js	function <b>set_portState</b> ( <b>newval</b> )
nodejs	function <b>set_portState</b> ( <b>newval</b> )
php	function <b>set_portState</b> ( <b>\$newval</b> )
cpp	int <b>set_portState</b> ( int <b>newval</b> )
m	-(int) <b>setPortState</b> : (int) <b>newval</b>
pas	function <b>set_portState</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_portState</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_portState</b> ( int <b>newval</b> )
java	int <b>set_portState</b> ( int <b>newval</b> )
py	def <b>set_portState</b> ( <b>newval</b> )
cmd	<b>YDigitalIO target set_portState newval</b>

This function has no effect on bits configured as input in `portDirection`.

**Parameters :**

**newval** an integer corresponding to the digital IO port state: bit 0 represents input 0, and so on

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**digitalio**→**set\_userdata()****YDigitalIO****digitalio**→**setUserData()****digitalio.set\_userdata()**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**digitalio→toggle\_bitState()digitalio.toggle\_bitState()****YDigitalIO**

Reverts a single bit of the I/O port.

js	function <b>toggle_bitState</b> ( <b>bitno</b> )
nodejs	function <b>toggle_bitState</b> ( <b>bitno</b> )
php	function <b>toggle_bitState</b> ( <b>\$bitno</b> )
cpp	int <b>toggle_bitState</b> ( int <b>bitno</b> )
m	-(int) <b>toggle_bitState</b> : (int) <b>bitno</b>
pas	function <b>toggle_bitState</b> ( <b>bitno</b> : LongInt): LongInt
vb	function <b>toggle_bitState</b> ( ) As Integer
cs	int <b>toggle_bitState</b> ( int <b>bitno</b> )
java	int <b>toggle_bitState</b> ( int <b>bitno</b> )
py	def <b>toggle_bitState</b> ( <b>bitno</b> )
cmd	YDigitalIO <b>target toggle_bitState bitno</b>

**Parameters :**

**bitno** the bit number; lowest bit has index 0

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**digitalio**→**wait\_async()****YDigitalIO**

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.

## 3.13. Display function interface

Yoctopuce display interface has been designed to easily show information and images. The device provides built-in multi-layer rendering. Layers can be drawn offline, individually, and freely moved on the display. It can also replay recorded sequences (animations).

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

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

### Global functions

#### yFindDisplay(func)

Retrieves a display for a given identifier.

#### yFirstDisplay()

Starts the enumeration of displays currently accessible.

### YDisplay methods

#### display→copyLayerContent(srcLayerId, dstLayerId)

Copies the whole content of a layer to another layer.

#### display→describe()

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

#### display→fade(brightness, duration)

Smoothly changes the brightness of the screen to produce a fade-in or fade-out effect.

#### display→get\_advertisedValue()

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

#### display→get\_brightness()

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

#### display→get\_displayHeight()

Returns the display height, in pixels.

#### display→get\_displayLayer(layerId)

Returns a YDisplayLayer object that can be used to draw on the specified layer.

#### display→get\_displayType()

Returns the display type: monochrome, gray levels or full color.

#### display→get\_displayWidth()

Returns the display width, in pixels.

#### display→get\_enabled()

Returns true if the screen is powered, false otherwise.

#### display→get\_errorMessage()

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

**display→get\_errorType()**

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

**display→get\_friendlyName()**

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

**display→get\_functionDescriptor()**

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

**display→get\_functionId()**

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

**display→get\_hardwareId()**

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

**display→get\_layerCount()**

Returns the number of available layers to draw on.

**display→get\_layerHeight()**

Returns the height of the layers to draw on, in pixels.

**display→get\_layerWidth()**

Returns the width of the layers to draw on, in pixels.

**display→get\_logicalName()**

Returns the logical name of the display.

**display→get\_module()**

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

**display→get\_module\_async(callback, context)**

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

**display→get\_orientation()**

Returns the currently selected display orientation.

**display→get\_startupSeq()**

Returns the name of the sequence to play when the displayed is powered on.

**display→get\_userData()**

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

**display→isOnline()**

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

**display→isOnline\_async(callback, context)**

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

**display→load(msValidity)**

Preloads the display cache with a specified validity duration.

**display→load\_async(msValidity, callback, context)**

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

**display→newSequence()**

Starts to record all display commands into a sequence, for later replay.

**display→nextDisplay()**

Continues the enumeration of displays started using `yFirstDisplay()`.

**display→pauseSequence(delay\_ms)**

Waits for a specified delay (in milliseconds) before playing next commands in current sequence.

**display→playSequence(sequenceName)**

Replays a display sequence previously recorded using `newSequence()` and `saveSequence()`.

**display→registerValueCallback(callback)**

### 3. Reference

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

#### **display→resetAll()**

Clears the display screen and resets all display layers to their default state.

#### **display→saveSequence(sequenceName)**

Stops recording display commands and saves the sequence into the specified file on the display internal memory.

#### **display→set\_brightness(newval)**

Changes the brightness of the display.

#### **display→set\_enabled(newval)**

Changes the power state of the display.

#### **display→set\_logicalName(newval)**

Changes the logical name of the display.

#### **display→set\_orientation(newval)**

Changes the display orientation.

#### **display→set\_startupSeq(newval)**

Changes the name of the sequence to play when the displayed is powered on.

#### **display→set\_userData(data)**

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

#### **display→stopSequence()**

Stops immediately any ongoing sequence replay.

#### **display→swapLayerContent(layerIdA, layerIdB)**

Swaps the whole content of two layers.

#### **display→upload(pathname, content)**

Uploads an arbitrary file (for instance a GIF file) to the display, to the specified full path name.

#### **display→wait\_async(callback, context)**

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

## YDisplay.FindDisplay() yFindDisplay()YDisplay.FindDisplay()

## YDisplay

Retrieves a display for a given identifier.

js	function <b>yFindDisplay</b> ( <b>func</b> )
nodejs	function <b>FindDisplay</b> ( <b>func</b> )
php	function <b>yFindDisplay</b> ( <b>\$func</b> )
cpp	YDisplay* <b>yFindDisplay</b> ( string <b>func</b> )
m	+(YDisplay*) <b>yFindDisplay</b> : (NSString*) <b>func</b>
pas	function <b>yFindDisplay</b> ( <b>func</b> : string): TYDisplay
vb	function <b>yFindDisplay</b> ( ByVal <b>func</b> As String) As YDisplay
cs	YDisplay <b>FindDisplay</b> ( string <b>func</b> )
java	YDisplay <b>FindDisplay</b> ( String <b>func</b> )
py	def <b>FindDisplay</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the display is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YDisplay.isOnline()` to test if the display is indeed online at a given time. In case of ambiguity when looking for a display 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 display

### Returns :

a YDisplay object allowing you to drive the display.

## YDisplay.FirstDisplay() yFirstDisplay()YDisplay.FirstDisplay()

YDisplay

Starts the enumeration of displays currently accessible.

js	function <b>yFirstDisplay</b> ( )
nodejs	function <b>FirstDisplay</b> ( )
php	function <b>yFirstDisplay</b> ( )
cpp	YDisplay* <b>yFirstDisplay</b> ( )
m	YDisplay* <b>yFirstDisplay</b> ( )
pas	function <b>yFirstDisplay</b> ( ): TYDisplay
vb	function <b>yFirstDisplay</b> ( ) As YDisplay
cs	YDisplay <b>FirstDisplay</b> ( )
java	YDisplay <b>FirstDisplay</b> ( )
py	def <b>FirstDisplay</b> ( )

Use the method `YDisplay.nextDisplay( )` to iterate on next displays.

**Returns :**

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



**display→copyLayerContent()  
display.copyLayerContent()**

## YDisplay

Copies the whole content of a layer to another layer.

js	function <b>copyLayerContent</b> ( <b>srcLayerId</b> , <b>dstLayerId</b> )
nodejs	function <b>copyLayerContent</b> ( <b>srcLayerId</b> , <b>dstLayerId</b> )
php	function <b>copyLayerContent</b> ( <b>\$srcLayerId</b> , <b>\$dstLayerId</b> )
c++	int <b>copyLayerContent</b> ( int <b>srcLayerId</b> , int <b>dstLayerId</b> )
m	-(int) <b>copyLayerContent</b> : (int) <b>srcLayerId</b> : (int) <b>dstLayerId</b>
pas	function <b>copyLayerContent</b> ( <b>srcLayerId</b> : LongInt, <b>dstLayerId</b> : LongInt): LongInt
vb	function <b>copyLayerContent</b> ( ) As Integer
cs	int <b>copyLayerContent</b> ( int <b>srcLayerId</b> , int <b>dstLayerId</b> )
java	int <b>copyLayerContent</b> ( int <b>srcLayerId</b> , int <b>dstLayerId</b> )
py	def <b>copyLayerContent</b> ( <b>srcLayerId</b> , <b>dstLayerId</b> )
cmd	YDisplay <b>target copyLayerContent</b> <b>srcLayerId</b> <b>dstLayerId</b>

The color and transparency of all the pixels from the destination layer are set to match the source pixels. This method only affects the displayed content, but does not change any property of the layer object. Note that layer 0 has no transparency support (it is always completely opaque).

### Parameters :

**srcLayerId** the identifier of the source layer (a number in range 0..layerCount-1)

**dstLayerId** the identifier of the destination layer (a number in range 0..layerCount-1)

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**display→describe()****display.describe()****YDisplay**

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

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

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(BadCustomName.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 display (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**display→fade()display.fade()****YDisplay**

Smoothly changes the brightness of the screen to produce a fade-in or fade-out effect.

js	function <b>fade</b> ( <b>brightness</b> , <b>duration</b> )
nodejs	function <b>fade</b> ( <b>brightness</b> , <b>duration</b> )
php	function <b>fade</b> ( <b>\$brightness</b> , <b>\$duration</b> )
cpp	int <b>fade</b> ( int <b>brightness</b> , int <b>duration</b> )
m	-(int) <b>fade</b> : (int) <b>brightness</b> : (int) <b>duration</b>
pas	function <b>fade</b> ( <b>brightness</b> : LongInt, <b>duration</b> : LongInt): LongInt
vb	function <b>fade</b> ( ) As Integer
cs	int <b>fade</b> ( int <b>brightness</b> , int <b>duration</b> )
java	int <b>fade</b> ( int <b>brightness</b> , int <b>duration</b> )
py	def <b>fade</b> ( <b>brightness</b> , <b>duration</b> )
cmd	YDisplay <b>target fade brightness duration</b>

**Parameters :**

- brightness** the new screen brightness
- duration** duration of the brightness transition, in milliseconds.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display**→**get\_advertisedValue()****YDisplay****display**→**advertisedValue()****display.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the display (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**display**→**get\_brightness()****YDisplay****display**→**brightness()****display.get\_brightness()**

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

js	function <b>get_brightness</b> ( )
nodejs	function <b>get_brightness</b> ( )
php	function <b>get_brightness</b> ( )
cpp	int <b>get_brightness</b> ( )
m	-(int) brightness
pas	function <b>get_brightness</b> ( ): LongInt
vb	function <b>get_brightness</b> ( ) As Integer
cs	int <b>get_brightness</b> ( )
java	int <b>get_brightness</b> ( )
py	def <b>get_brightness</b> ( )
cmd	YDisplay <b>target</b> <b>get_brightness</b>

**Returns :**

an integer corresponding to the luminosity of the module informative leds (from 0 to 100)

On failure, throws an exception or returns Y\_BRIGHTNESS\_INVALID.

**display**→**get\_displayHeight()****YDisplay****display**→**displayHeight()****display.get\_displayHeight()**

Returns the display height, in pixels.

js	function <b>get_displayHeight</b> ( )
nodejs	function <b>get_displayHeight</b> ( )
php	function <b>get_displayHeight</b> ( )
cpp	int <b>get_displayHeight</b> ( )
m	-(int) displayHeight
pas	function <b>get_displayHeight</b> ( ): LongInt
vb	function <b>get_displayHeight</b> ( ) As Integer
cs	int <b>get_displayHeight</b> ( )
java	int <b>get_displayHeight</b> ( )
py	def <b>get_displayHeight</b> ( )
cmd	YDisplay <b>target</b> <b>get_displayHeight</b>

**Returns :**

an integer corresponding to the display height, in pixels

On failure, throws an exception or returns Y\_DISPLAYHEIGHT\_INVALID.

**display**→**get\_displayLayer()****YDisplay****display**→**displayLayer()****display.get\_displayLayer()**

Returns a YDisplayLayer object that can be used to draw on the specified layer.

js	function <b>get_displayLayer</b> ( <b>layerId</b> )
nodejs	function <b>get_displayLayer</b> ( <b>layerId</b> )
php	function <b>get_displayLayer</b> ( <b>\$layerId</b> )
cpp	YDisplayLayer* <b>get_displayLayer</b> ( unsigned <b>layerId</b> )
m	-(YDisplayLayer*) displayLayer : (unsigned) <b>layerId</b>
vb	function <b>get_displayLayer</b> ( ) As YDisplayLayer
cs	YDisplayLayer <b>get_displayLayer</b> ( int <b>layerId</b> )
java	synchronized YDisplayLayer <b>get_displayLayer</b> ( int <b>layerId</b> )
py	def <b>get_displayLayer</b> ( <b>layerId</b> )

The content is displayed only when the layer is active on the screen (and not masked by other overlapping layers).

**Parameters :**

**layerId** the identifier of the layer (a number in range 0..layerCount-1)

**Returns :**

an YDisplayLayer object

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

**display**→**get\_displayType()****YDisplay****display**→**displayType()****display.get\_displayType()**

Returns the display type: monochrome, gray levels or full color.

js	function <b>get_displayType</b> ( )
nodejs	function <b>get_displayType</b> ( )
php	function <b>get_displayType</b> ( )
cpp	Y_DISPLAYTYPE_enum <b>get_displayType</b> ( )
m	-(Y_DISPLAYTYPE_enum) displayType
pas	function <b>get_displayType</b> ( ): Integer
vb	function <b>get_displayType</b> ( ) As Integer
cs	int <b>get_displayType</b> ( )
java	int <b>get_displayType</b> ( )
py	def <b>get_displayType</b> ( )
cmd	YDisplay <b>target</b> <b>get_displayType</b>

**Returns :**

a value among Y\_DISPLAYTYPE\_MONO, Y\_DISPLAYTYPE\_GRAY and Y\_DISPLAYTYPE\_RGB corresponding to the display type: monochrome, gray levels or full color

On failure, throws an exception or returns Y\_DISPLAYTYPE\_INVALID.



**display**→**get\_displayWidth()****YDisplay****display**→**displayWidth()****display.get\_displayWidth()**

Returns the display width, in pixels.

js	function <b>get_displayWidth</b> ( )
nodejs	function <b>get_displayWidth</b> ( )
php	function <b>get_displayWidth</b> ( )
cpp	int <b>get_displayWidth</b> ( )
m	-(int) displayWidth
pas	function <b>get_displayWidth</b> ( ): LongInt
vb	function <b>get_displayWidth</b> ( ) As Integer
cs	int <b>get_displayWidth</b> ( )
java	int <b>get_displayWidth</b> ( )
py	def <b>get_displayWidth</b> ( )
cmd	YDisplay <b>target</b> <b>get_displayWidth</b>

**Returns :**

an integer corresponding to the display width, in pixels

On failure, throws an exception or returns Y\_DISPLAYWIDTH\_INVALID.

**display**→**get\_enabled()****YDisplay****display**→**enabled()****display.get\_enabled()**

Returns true if the screen is powered, false otherwise.

js	function <b>get_enabled</b> ( )
nodejs	function <b>get_enabled</b> ( )
php	function <b>get_enabled</b> ( )
cpp	Y_ENABLED_enum <b>get_enabled</b> ( )
m	-(Y_ENABLED_enum) enabled
pas	function <b>get_enabled</b> ( ): Integer
vb	function <b>get_enabled</b> ( ) As Integer
cs	int <b>get_enabled</b> ( )
java	int <b>get_enabled</b> ( )
py	def <b>get_enabled</b> ( )
cmd	YDisplay <b>target</b> <b>get_enabled</b>

**Returns :**

either Y\_ENABLED\_FALSE or Y\_ENABLED\_TRUE, according to true if the screen is powered, false otherwise

On failure, throws an exception or returns Y\_ENABLED\_INVALID.

**display**→**get\_errorMessage()****YDisplay****display**→**errorMessage()****display.get\_errorMessage()**

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

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the display object

**display**→**get\_errorType()****YDisplay****display**→**errorType()****display.get\_errorType()**

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

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the display object

**display**→**get\_friendlyName()****YDisplay****display**→**friendlyName()****display.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the display using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**display→get\_functionDescriptor()**  
**display→functionDescriptor()**  
**display.get\_functionDescriptor()**

**YDisplay**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**display**→**get\_functionId()****YDisplay****display**→**functionId()****display.get\_functionId()**

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

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

For example `relay1`

**Returns :**

a string that identifies the display (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**display**→**get\_hardwareId()****YDisplay****display**→**hardwareId()****display.get\_hardwareId()**

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

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

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

**Returns :**

a string that uniquely identifies the display (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.



**display**→**get\_layerCount()****YDisplay****display**→**layerCount()****display.get\_layerCount()**

Returns the number of available layers to draw on.

js	function <b>get_layerCount</b> ( )
nodejs	function <b>get_layerCount</b> ( )
php	function <b>get_layerCount</b> ( )
cpp	int <b>get_layerCount</b> ( )
m	-(int) layerCount
pas	function <b>get_layerCount</b> ( ): LongInt
vb	function <b>get_layerCount</b> ( ) As Integer
cs	int <b>get_layerCount</b> ( )
java	int <b>get_layerCount</b> ( )
py	def <b>get_layerCount</b> ( )
cmd	YDisplay <b>target</b> <b>get_layerCount</b>

**Returns :**

an integer corresponding to the number of available layers to draw on

On failure, throws an exception or returns Y\_LAYERCOUNT\_INVALID.

**display**→**get\_layerHeight()****YDisplay****display**→**layerHeight()****display.get\_layerHeight()**

Returns the height of the layers to draw on, in pixels.

js	function <b>get_layerHeight</b> ( )
nodejs	function <b>get_layerHeight</b> ( )
php	function <b>get_layerHeight</b> ( )
cpp	int <b>get_layerHeight</b> ( )
m	-(int) layerHeight
pas	function <b>get_layerHeight</b> ( ): LongInt
vb	function <b>get_layerHeight</b> ( ) As Integer
cs	int <b>get_layerHeight</b> ( )
java	int <b>get_layerHeight</b> ( )
py	def <b>get_layerHeight</b> ( )
cmd	YDisplay <b>target</b> <b>get_layerHeight</b>

**Returns :**

an integer corresponding to the height of the layers to draw on, in pixels

On failure, throws an exception or returns Y\_LAYERHEIGHT\_INVALID.

**display**→**get\_layerWidth()****YDisplay****display**→**layerWidth()****display.get\_layerWidth()**

Returns the width of the layers to draw on, in pixels.

js	function <b>get_layerWidth</b> ( )
nodejs	function <b>get_layerWidth</b> ( )
php	function <b>get_layerWidth</b> ( )
cpp	int <b>get_layerWidth</b> ( )
m	-(int) layerWidth
pas	function <b>get_layerWidth</b> ( ): LongInt
vb	function <b>get_layerWidth</b> ( ) As Integer
cs	int <b>get_layerWidth</b> ( )
java	int <b>get_layerWidth</b> ( )
py	def <b>get_layerWidth</b> ( )
cmd	YDisplay <b>target</b> <b>get_layerWidth</b>

**Returns :**

an integer corresponding to the width of the layers to draw on, in pixels

On failure, throws an exception or returns Y\_LAYERWIDTH\_INVALID.

**display**→**get\_logicalName()****YDisplay****display**→**logicalName()****display.get\_logicalName()**

Returns the logical name of the display.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YDisplay <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the display. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**display**→**get\_module()****YDisplay****display**→**module()****display.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**display→get\_module\_async()****YDisplay****display→module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**display**→**get\_orientation()****YDisplay****display**→**orientation()****display.get\_orientation()**

Returns the currently selected display orientation.

js	function <b>get_orientation</b> ( )
nodejs	function <b>get_orientation</b> ( )
php	function <b>get_orientation</b> ( )
cpp	Y_ORIENTATION_enum <b>get_orientation</b> ( )
m	-(Y_ORIENTATION_enum) orientation
pas	function <b>get_orientation</b> ( ): Integer
vb	function <b>get_orientation</b> ( ) As Integer
cs	int <b>get_orientation</b> ( )
java	int <b>get_orientation</b> ( )
py	def <b>get_orientation</b> ( )
cmd	YDisplay <b>target</b> <b>get_orientation</b>

**Returns :**

a value among Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP, Y\_ORIENTATION\_RIGHT and Y\_ORIENTATION\_DOWN corresponding to the currently selected display orientation

On failure, throws an exception or returns Y\_ORIENTATION\_INVALID.

**display**→**get\_startupSeq()****YDisplay****display**→**startupSeq()****display.get\_startupSeq()**

Returns the name of the sequence to play when the displayed is powered on.

js	function <b>get_startupSeq</b> ( )
nodejs	function <b>get_startupSeq</b> ( )
php	function <b>get_startupSeq</b> ( )
cpp	string <b>get_startupSeq</b> ( )
m	-(NSString*) startupSeq
pas	function <b>get_startupSeq</b> ( ): string
vb	function <b>get_startupSeq</b> ( ) As String
cs	string <b>get_startupSeq</b> ( )
java	String <b>get_startupSeq</b> ( )
py	def <b>get_startupSeq</b> ( )
cmd	YDisplay <b>target</b> <b>get_startupSeq</b>

**Returns :**

a string corresponding to the name of the sequence to play when the displayed is powered on

On failure, throws an exception or returns Y\_STARTUPSEQ\_INVALID.



**display**→**get\_userData()****YDisplay****display**→**userData()****display.get\_userData()**

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

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

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.

**display**→**isOnline()****display.isOnline()****YDisplay**

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

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

If there is a cached value for the display 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 display.

**Returns :**

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

**display→isOnline\_async()****YDisplay**

Checks if the display 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 display 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.

**display**→**load()****display.load()****YDisplay**

Preloads the display cache with a specified validity duration.

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

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 traffic 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.

**display→load\_async()****YDisplay**

Preloads the display 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 traffic 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.

**display**→**newSequence()****display.newSequence()****YDisplay**

Starts to record all display commands into a sequence, for later replay.

js	function <b>newSequence</b> ( )
nodejs	function <b>newSequence</b> ( )
php	function <b>newSequence</b> ( )
cpp	int <b>newSequence</b> ( )
m	-(int) <b>newSequence</b>
pas	function <b>newSequence</b> ( ): LongInt
vb	function <b>newSequence</b> ( ) As Integer
cs	int <b>newSequence</b> ( )
java	int <b>newSequence</b> ( )
py	def <b>newSequence</b> ( )
cmd	YDisplay <b>target newSequence</b>

The name used to store the sequence is specified when calling `saveSequence( )`, once the recording is complete.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display**→**nextDisplay()****display.nextDisplay()****YDisplay**

Continues the enumeration of displays started using `yFirstDisplay()`.

js	function <b>nextDisplay</b> ( )
nodejs	function <b>nextDisplay</b> ( )
php	function <b>nextDisplay</b> ( )
cpp	YDisplay * <b>nextDisplay</b> ( )
m	-(YDisplay*) <b>nextDisplay</b>
pas	function <b>nextDisplay</b> ( ): TYDisplay
vb	function <b>nextDisplay</b> ( ) As YDisplay
cs	YDisplay <b>nextDisplay</b> ( )
java	YDisplay <b>nextDisplay</b> ( )
py	def <b>nextDisplay</b> ( )

**Returns :**

a pointer to a `YDisplay` object, corresponding to a display currently online, or a `null` pointer if there are no more displays to enumerate.

**display→pauseSequence()display.pauseSequence()****YDisplay**

Waits for a specified delay (in milliseconds) before playing next commands in current sequence.

js	function <b>pauseSequence</b> ( <b>delay_ms</b> )
nodejs	function <b>pauseSequence</b> ( <b>delay_ms</b> )
php	function <b>pauseSequence</b> ( <b>\$delay_ms</b> )
cpp	int <b>pauseSequence</b> ( int <b>delay_ms</b> )
m	-(int) <b>pauseSequence</b> : (int) <b>delay_ms</b>
pas	function <b>pauseSequence</b> ( <b>delay_ms</b> : LongInt): LongInt
vb	function <b>pauseSequence</b> ( ) As Integer
cs	int <b>pauseSequence</b> ( int <b>delay_ms</b> )
java	int <b>pauseSequence</b> ( int <b>delay_ms</b> )
py	def <b>pauseSequence</b> ( <b>delay_ms</b> )
cmd	YDisplay <b>target</b> <b>pauseSequence</b> <b>delay_ms</b>

This method can be used while recording a display sequence, to insert a timed wait in the sequence (without any immediate effect). It can also be used dynamically while playing a pre-recorded sequence, to suspend or resume the execution of the sequence. To cancel a delay, call the same method with a zero delay.

**Parameters :**

**delay\_ms** the duration to wait, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**display→playSequence()****display.playSequence()****YDisplay**

Replays a display sequence previously recorded using `newSequence()` and `saveSequence()`.

js	function <b>playSequence</b> ( <b>sequenceName</b> )
nodejs	function <b>playSequence</b> ( <b>sequenceName</b> )
php	function <b>playSequence</b> ( <b>\$sequenceName</b> )
cpp	int <b>playSequence</b> ( string <b>sequenceName</b> )
m	-(int) <b>playSequence</b> : (NSString*) <b>sequenceName</b>
pas	function <b>playSequence</b> ( <b>sequenceName</b> : string): LongInt
vb	function <b>playSequence</b> ( ) As Integer
cs	int <b>playSequence</b> ( string <b>sequenceName</b> )
java	int <b>playSequence</b> ( String <b>sequenceName</b> )
py	def <b>playSequence</b> ( <b>sequenceName</b> )
cmd	YDisplay <b>target playSequence</b> <b>sequenceName</b>

**Parameters :**

**sequenceName** the name of the newly created sequence

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## display→registerValueCallback() display.registerValueCallback()

YDisplay

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YDisplayValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YDisplayValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYDisplayValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**display→resetAll()display.resetAll()****YDisplay**

Clears the display screen and resets all display layers to their default state.

js	function <b>resetAll</b> ( )
nodejs	function <b>resetAll</b> ( )
php	function <b>resetAll</b> ( )
cpp	int <b>resetAll</b> ( )
m	-(int) <b>resetAll</b>
pas	function <b>resetAll</b> ( ): LongInt
vb	function <b>resetAll</b> ( ) As Integer
cs	int <b>resetAll</b> ( )
java	int <b>resetAll</b> ( )
py	def <b>resetAll</b> ( )
cmd	YDisplay <b>target</b> <b>resetAll</b>

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display→saveSequence()display.saveSequence()****YDisplay**

Stops recording display commands and saves the sequence into the specified file on the display internal memory.

js	function <b>saveSequence</b> ( <b>sequenceName</b> )
nodejs	function <b>saveSequence</b> ( <b>sequenceName</b> )
php	function <b>saveSequence</b> ( <b>\$sequenceName</b> )
cpp	int <b>saveSequence</b> ( string <b>sequenceName</b> )
m	-(int) <b>saveSequence</b> : (NSString*) <b>sequenceName</b>
pas	function <b>saveSequence</b> ( <b>sequenceName</b> : string): LongInt
vb	function <b>saveSequence</b> ( ) As Integer
cs	int <b>saveSequence</b> ( string <b>sequenceName</b> )
java	int <b>saveSequence</b> ( String <b>sequenceName</b> )
py	def <b>saveSequence</b> ( <b>sequenceName</b> )
cmd	YDisplay <b>target saveSequence</b> <b>sequenceName</b>

The sequence can be later replayed using `playSequence ( )`.

**Parameters :**

**sequenceName** the name of the newly created sequence

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display**→**set\_brightness()****YDisplay****display**→**setBrightness()****display.set\_brightness()**

Changes the brightness of the display.

js	function <b>set_brightness</b> ( <b>newval</b> )
nodejs	function <b>set_brightness</b> ( <b>newval</b> )
php	function <b>set_brightness</b> ( <b>\$newval</b> )
cpp	int <b>set_brightness</b> ( int <b>newval</b> )
m	-(int) <b>setBrightness</b> : (int) <b>newval</b>
pas	function <b>set_brightness</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_brightness</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_brightness</b> ( int <b>newval</b> )
java	int <b>set_brightness</b> ( int <b>newval</b> )
py	def <b>set_brightness</b> ( <b>newval</b> )
cmd	YDisplay <b>target</b> <b>set_brightness</b> <b>newval</b>

The parameter is a value between 0 and 100. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

**Parameters :**

**newval** an integer corresponding to the brightness of the display

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display**→**set\_enabled()****YDisplay****display**→**setEnabled()****display.set\_enabled()**

Changes the power state of the display.

js	function <b>set_enabled</b> ( <b>newval</b> )
nodejs	function <b>set_enabled</b> ( <b>newval</b> )
php	function <b>set_enabled</b> ( <b>\$newval</b> )
cpp	int <b>set_enabled</b> ( Y_ENABLED_enum <b>newval</b> )
m	-(int) <b>setEnabled</b> : (Y_ENABLED_enum) <b>newval</b>
pas	function <b>set_enabled</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_enabled</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_enabled</b> ( int <b>newval</b> )
java	int <b>set_enabled</b> ( int <b>newval</b> )
py	def <b>set_enabled</b> ( <b>newval</b> )
cmd	YDisplay <b>target set_enabled newval</b>

**Parameters :**

**newval** either Y\_ENABLED\_FALSE or Y\_ENABLED\_TRUE, according to the power state of the display

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display**→**set\_logicalName()****YDisplay****display**→**setLogicalName()****display.set\_logicalName()**

Changes the logical name of the display.

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

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 display.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**display**→**set\_orientation()****YDisplay****display**→**setOrientation()****display.set\_orientation()**

Changes the display orientation.

js	function <b>set_orientation</b> ( <b>newval</b> )
nodejs	function <b>set_orientation</b> ( <b>newval</b> )
php	function <b>set_orientation</b> ( <b>\$newval</b> )
cpp	int <b>set_orientation</b> ( Y_ORIENTATION_enum <b>newval</b> )
m	-(int) setOrientation : (Y_ORIENTATION_enum) <b>newval</b>
pas	function <b>set_orientation</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_orientation</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_orientation</b> ( int <b>newval</b> )
java	int <b>set_orientation</b> ( int <b>newval</b> )
py	def <b>set_orientation</b> ( <b>newval</b> )
cmd	YDisplay <b>target set_orientation newval</b>

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

**Parameters :**

**newval** a value among Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP, Y\_ORIENTATION\_RIGHT and Y\_ORIENTATION\_DOWN corresponding to the display orientation

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**display**→**set\_startupSeq()****YDisplay****display**→**setStartupSeq()****display.set\_startupSeq()**

Changes the name of the sequence to play when the displayed is powered on.

js	function <b>set_startupSeq</b> ( <b>newval</b> )
nodejs	function <b>set_startupSeq</b> ( <b>newval</b> )
php	function <b>set_startupSeq</b> ( <b>\$newval</b> )
cpp	int <b>set_startupSeq</b> ( const string& <b>newval</b> )
m	-(int) setStartupSeq : (NSString*) <b>newval</b>
pas	function <b>set_startupSeq</b> ( <b>newval</b> : string): integer
vb	function <b>set_startupSeq</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_startupSeq</b> ( string <b>newval</b> )
java	int <b>set_startupSeq</b> ( String <b>newval</b> )
py	def <b>set_startupSeq</b> ( <b>newval</b> )
cmd	YDisplay <b>target set_startupSeq newval</b>

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

**Parameters :**

**newval** a string corresponding to the name of the sequence to play when the displayed is powered on

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display**→**set\_userData()****YDisplay****display**→**setUserData()****display.set\_userData()**

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

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

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

**display→stopSequence()display.stopSequence()****YDisplay**

Stops immediately any ongoing sequence replay.

js	function <b>stopSequence</b> ( )
nodejs	function <b>stopSequence</b> ( )
php	function <b>stopSequence</b> ( )
cpp	int <b>stopSequence</b> ( )
m	-(int) <b>stopSequence</b>
pas	function <b>stopSequence</b> ( ): LongInt
vb	function <b>stopSequence</b> ( ) As Integer
cs	int <b>stopSequence</b> ( )
java	int <b>stopSequence</b> ( )
py	def <b>stopSequence</b> ( )
cmd	YDisplay <b>target</b> <b>stopSequence</b>

The display is left as is.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

Swaps the whole content of two layers.

js	function <b>swapLayerContent</b> ( <b>layerIdA</b> , <b>layerIdB</b> )
nodejs	function <b>swapLayerContent</b> ( <b>layerIdA</b> , <b>layerIdB</b> )
php	function <b>swapLayerContent</b> ( <b>\$layerIdA</b> , <b>\$layerIdB</b> )
cpp	int <b>swapLayerContent</b> ( int <b>layerIdA</b> , int <b>layerIdB</b> )
m	-(int) <b>swapLayerContent</b> : (int) <b>layerIdA</b> : (int) <b>layerIdB</b>
pas	function <b>swapLayerContent</b> ( <b>layerIdA</b> : LongInt, <b>layerIdB</b> : LongInt): LongInt
vb	function <b>swapLayerContent</b> ( ) As Integer
cs	int <b>swapLayerContent</b> ( int <b>layerIdA</b> , int <b>layerIdB</b> )
java	int <b>swapLayerContent</b> ( int <b>layerIdA</b> , int <b>layerIdB</b> )
py	def <b>swapLayerContent</b> ( <b>layerIdA</b> , <b>layerIdB</b> )
cmd	YDisplay <b>target swapLayerContent</b> <b>layerIdA layerIdB</b>

The color and transparency of all the pixels from the two layers are swapped. This method only affects the displayed content, but does not change any property of the layer objects. In particular, the visibility of each layer stays unchanged. When used between one hidden layer and a visible layer, this method makes it possible to easily implement double-buffering. Note that layer 0 has no transparency support (it is always completely opaque).

### Parameters :

**layerIdA** the first layer (a number in range 0..layerCount-1)  
**layerIdB** the second layer (a number in range 0..layerCount-1)

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**display→upload()display.upload()****YDisplay**

Uploads an arbitrary file (for instance a GIF file) to the display, to the specified full path name.

js	function <b>upload</b> ( <b>pathname</b> , <b>content</b> )
nodejs	function <b>upload</b> ( <b>pathname</b> , <b>content</b> )
php	function <b>upload</b> ( <b>\$pathname</b> , <b>\$content</b> )
cpp	int <b>upload</b> ( string <b>pathname</b> , string <b>content</b> )
m	-(int) <b>upload</b> : (NSString*) <b>pathname</b> : (NSData*) <b>content</b>
pas	function <b>upload</b> ( <b>pathname</b> : string, <b>content</b> : TByteArray): LongInt
vb	procedure <b>upload</b> ( )
cs	int <b>upload</b> ( string <b>pathname</b> )
java	int <b>upload</b> ( String <b>pathname</b> )
py	def <b>upload</b> ( <b>pathname</b> , <b>content</b> )
cmd	YDisplay <b>target upload</b> <b>pathname content</b>

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.

**display→wait\_async()****YDisplay**

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.

## 3.14. DisplayLayer object interface

A DisplayLayer is an image layer containing objects to display (bitmaps, text, etc.). The content is displayed only when the layer is active on the screen (and not masked by other overlapping layers).

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

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

### YDisplayLayer methods

#### displaylayer→clear()

Erases the whole content of the layer (makes it fully transparent).

#### displaylayer→clearConsole()

Blanks the console area within console margins, and resets the console pointer to the upper left corner of the console.

#### displaylayer→consoleOut(text)

Outputs a message in the console area, and advances the console pointer accordingly.

#### displaylayer→drawBar(x1, y1, x2, y2)

Draws a filled rectangular bar at a specified position.

#### displaylayer→drawBitmap(x, y, w, bitmap, bgcolor)

Draws a bitmap at the specified position.

#### displaylayer→drawCircle(x, y, r)

Draws an empty circle at a specified position.

#### displaylayer→drawDisc(x, y, r)

Draws a filled disc at a given position.

#### displaylayer→drawImage(x, y, imagename)

Draws a GIF image at the specified position.

#### displaylayer→drawPixel(x, y)

Draws a single pixel at the specified position.

#### displaylayer→drawRect(x1, y1, x2, y2)

Draws an empty rectangle at a specified position.

#### displaylayer→drawText(x, y, anchor, text)

Draws a text string at the specified position.

#### displaylayer→get\_display()

Gets parent YDisplay.

#### displaylayer→get\_displayHeight()

Returns the display height, in pixels.

#### displaylayer→get\_displayWidth()

Returns the display width, in pixels.

**displaylayer→get\_layerHeight()**

Returns the height of the layers to draw on, in pixels.

**displaylayer→get\_layerWidth()**

Returns the width of the layers to draw on, in pixels.

**displaylayer→hide()**

Hides the layer.

**displaylayer→lineTo(x, y)**

Draws a line from current drawing pointer position to the specified position.

**displaylayer→moveTo(x, y)**

Moves the drawing pointer of this layer to the specified position.

**displaylayer→reset()**

Reverts the layer to its initial state (fully transparent, default settings).

**displaylayer→selectColorPen(color)**

Selects the pen color for all subsequent drawing functions, including text drawing.

**displaylayer→selectEraser()**

Selects an eraser instead of a pen for all subsequent drawing functions, except for text drawing and bitmap copy functions.

**displaylayer→selectFont(fontname)**

Selects a font to use for the next text drawing functions, by providing the name of the font file.

**displaylayer→selectGrayPen(graylevel)**

Selects the pen gray level for all subsequent drawing functions, including text drawing.

**displaylayer→setAntialiasingMode(mode)**

Enables or disables anti-aliasing for drawing oblique lines and circles.

**displaylayer→setConsoleBackground(bgcol)**

Sets up the background color used by the `clearConsole` function and by the console scrolling feature.

**displaylayer→setConsoleMargins(x1, y1, x2, y2)**

Sets up display margins for the `consoleOut` function.

**displaylayer→setConsoleWordWrap(wordwrap)**

Sets up the wrapping behaviour used by the `consoleOut` function.

**displaylayer→setLayerPosition(x, y, scrollTime)**

Sets the position of the layer relative to the display upper left corner.

**displaylayer→unhide()**

Shows the layer.



**displaylayer→clear()displaylayer.clear()****YDisplayLayer**

Erases the whole content of the layer (makes it fully transparent).

js	function <b>clear</b> ( )
nodejs	function <b>clear</b> ( )
php	function <b>clear</b> ( )
cpp	int <b>clear</b> ( )
m	-(int) <b>clear</b>
pas	function <b>clear</b> ( ): LongInt
vb	function <b>clear</b> ( ) As Integer
cs	int <b>clear</b> ( )
java	int <b>clear</b> ( )
py	def <b>clear</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>clear</b>

This method does not change any other attribute of the layer. To reinitialize the layer attributes to defaults settings, use the method `reset( )` instead.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→clearConsole()**  
**displaylayer.clearConsole()****YDisplayLayer**

Blanks the console area within console margins, and resets the console pointer to the upper left corner of the console.

js	function <b>clearConsole</b> ( )
nodejs	function <b>clearConsole</b> ( )
php	function <b>clearConsole</b> ( )
c++	int <b>clearConsole</b> ( )
m	-(int) <b>clearConsole</b>
pas	function <b>clearConsole</b> ( ): LongInt
vb	function <b>clearConsole</b> ( ) As Integer
cs	int <b>clearConsole</b> ( )
java	int <b>clearConsole</b> ( )
py	def <b>clearConsole</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>clearConsole</b>

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→consoleOut()displaylayer.consoleOut()****YDisplayLayer**

Outputs a message in the console area, and advances the console pointer accordingly.

js	function <b>consoleOut</b> ( <b>text</b> )
nodejs	function <b>consoleOut</b> ( <b>text</b> )
php	function <b>consoleOut</b> ( <b>\$text</b> )
cpp	int <b>consoleOut</b> ( string <b>text</b> )
m	-(int) <b>consoleOut</b> : (NSString*) <b>text</b>
pas	function <b>consoleOut</b> ( <b>text</b> : string): LongInt
vb	function <b>consoleOut</b> ( ) As Integer
cs	int <b>consoleOut</b> ( string <b>text</b> )
java	int <b>consoleOut</b> ( String <b>text</b> )
py	def <b>consoleOut</b> ( <b>text</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>consoleOut</b> <b>text</b>

The console pointer position is automatically moved to the beginning of the next line when a newline character is met, or when the right margin is hit. When the new text to display extends below the lower margin, the console area is automatically scrolled up.

**Parameters :**

**text** the message to display

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→drawBar()displaylayer.drawBar()****YDisplayLayer**

Draws a filled rectangular bar at a specified position.

js	function <b>drawBar</b> ( <b>x1</b> , <b>y1</b> , <b>x2</b> , <b>y2</b> )
nodejs	function <b>drawBar</b> ( <b>x1</b> , <b>y1</b> , <b>x2</b> , <b>y2</b> )
php	function <b>drawBar</b> ( <b>\$x1</b> , <b>\$y1</b> , <b>\$x2</b> , <b>\$y2</b> )
cpp	int <b>drawBar</b> ( int <b>x1</b> , int <b>y1</b> , int <b>x2</b> , int <b>y2</b> )
m	-(int) <b>drawBar</b> : (int) <b>x1</b> : (int) <b>y1</b> : (int) <b>x2</b> : (int) <b>y2</b>
pas	function <b>drawBar</b> ( <b>x1</b> : LongInt, <b>y1</b> : LongInt, <b>x2</b> : LongInt, <b>y2</b> : LongInt): LongInt
vb	function <b>drawBar</b> ( ) As Integer
cs	int <b>drawBar</b> ( int <b>x1</b> , int <b>y1</b> , int <b>x2</b> , int <b>y2</b> )
java	int <b>drawBar</b> ( int <b>x1</b> , int <b>y1</b> , int <b>x2</b> , int <b>y2</b> )
py	def <b>drawBar</b> ( <b>x1</b> , <b>y1</b> , <b>x2</b> , <b>y2</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>drawBar</b> <b>x1</b> <b>y1</b> <b>x2</b> <b>y2</b>

**Parameters :**

- x1** the distance from left of layer to the left border of the rectangle, in pixels
- y1** the distance from top of layer to the top border of the rectangle, in pixels
- x2** the distance from left of layer to the right border of the rectangle, in pixels
- y2** the distance from top of layer to the bottom border of the rectangle, in pixels

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## displaylayer→drawBitmap() displaylayer.drawBitmap()

YDisplayLayer

Draws a bitmap at the specified position.

js	function <b>drawBitmap</b> ( <b>x</b> , <b>y</b> , <b>w</b> , <b>bitmap</b> , <b>bgcol</b> )
nodejs	function <b>drawBitmap</b> ( <b>x</b> , <b>y</b> , <b>w</b> , <b>bitmap</b> , <b>bgcol</b> )
php	function <b>drawBitmap</b> ( <b>\$x</b> , <b>\$y</b> , <b>\$w</b> , <b>\$bitmap</b> , <b>\$bgcol</b> )
cpp	int <b>drawBitmap</b> ( int <b>x</b> , int <b>y</b> , int <b>w</b> , string <b>bitmap</b> , int <b>bgcol</b> )
m	-(int) <b>drawBitmap</b> : (int) <b>x</b> : (int) <b>y</b> : (int) <b>w</b> : (NSData*) <b>bitmap</b> : (int) <b>bgcol</b>
pas	function <b>drawBitmap</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt, <b>w</b> : LongInt, <b>bitmap</b> : TByteArray, <b>bgcol</b> : LongInt): LongInt
vb	procedure <b>drawBitmap</b> ( )
cs	int <b>drawBitmap</b> ( int <b>x</b> , int <b>y</b> , int <b>w</b> , int <b>bgcol</b> )
java	int <b>drawBitmap</b> ( int <b>x</b> , int <b>y</b> , int <b>w</b> , int <b>bgcol</b> )
py	def <b>drawBitmap</b> ( <b>x</b> , <b>y</b> , <b>w</b> , <b>bitmap</b> , <b>bgcol</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>drawBitmap</b> <b>x y w bitmap bgcol</b>

The bitmap is provided as a binary object, where each pixel maps to a bit, from left to right and from top to bottom. The most significant bit of each byte maps to the leftmost pixel, and the least significant bit maps to the rightmost pixel. Bits set to 1 are drawn using the layer selected pen color. Bits set to 0 are drawn using the specified background gray level, unless -1 is specified, in which case they are not drawn at all (as if transparent).

### Parameters :

- x** the distance from left of layer to the left of the bitmap, in pixels
- y** the distance from top of layer to the top of the bitmap, in pixels
- w** the width of the bitmap, in pixels
- bitmap** a binary object
- bgcol** the background gray level to use for zero bits (0 = black, 255 = white), or -1 to leave the pixels unchanged

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→drawCircle()displaylayer.drawCircle()**

## YDisplayLayer

Draws an empty circle at a specified position.

js	function <b>drawCircle</b> ( <b>x</b> , <b>y</b> , <b>r</b> )
nodejs	function <b>drawCircle</b> ( <b>x</b> , <b>y</b> , <b>r</b> )
php	function <b>drawCircle</b> ( <b>\$x</b> , <b>\$y</b> , <b>\$r</b> )
cpp	int <b>drawCircle</b> ( int <b>x</b> , int <b>y</b> , int <b>r</b> )
m	-(int) <b>drawCircle</b> : (int) <b>x</b> : (int) <b>y</b> : (int) <b>r</b>
pas	function <b>drawCircle</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt, <b>r</b> : LongInt): LongInt
vb	function <b>drawCircle</b> ( ) As Integer
cs	int <b>drawCircle</b> ( int <b>x</b> , int <b>y</b> , int <b>r</b> )
java	int <b>drawCircle</b> ( int <b>x</b> , int <b>y</b> , int <b>r</b> )
py	def <b>drawCircle</b> ( <b>x</b> , <b>y</b> , <b>r</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>drawCircle</b> <b>x y r</b>

### Parameters :

- x** the distance from left of layer to the center of the circle, in pixels
- y** the distance from top of layer to the center of the circle, in pixels
- r** the radius of the circle, in pixels

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

### Parameters :

### Returns :

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

**displaylayer→drawImage()displaylayer.drawImage()**

## YDisplayLayer

Draws a GIF image at the specified position.

js	function <b>drawImage</b> ( <b>x</b> , <b>y</b> , <b>imagename</b> )
nodejs	function <b>drawImage</b> ( <b>x</b> , <b>y</b> , <b>imagename</b> )
php	function <b>drawImage</b> ( <b>\$x</b> , <b>\$y</b> , <b>\$imagename</b> )
cpp	int <b>drawImage</b> ( int <b>x</b> , int <b>y</b> , string <b>imagename</b> )
m	-(int) <b>drawImage</b> : (int) <b>x</b> : (int) <b>y</b> : (NSString*) <b>imagename</b>
pas	function <b>drawImage</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt, <b>imagename</b> : string): LongInt
vb	function <b>drawImage</b> ( ) As Integer
cs	int <b>drawImage</b> ( int <b>x</b> , int <b>y</b> , string <b>imagename</b> )
java	int <b>drawImage</b> ( int <b>x</b> , int <b>y</b> , String <b>imagename</b> )
py	def <b>drawImage</b> ( <b>x</b> , <b>y</b> , <b>imagename</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>drawImage</b> <b>x</b> <b>y</b> <b>imagename</b>

The GIF image must have been previously uploaded to the device built-in memory. If you experience problems using an image file, check the device logs for any error message such as missing image file or bad image file format.

### Parameters :

<b>x</b>	the distance from left of layer to the left of the image, in pixels
<b>y</b>	the distance from top of layer to the top of the image, in pixels
<b>imagename</b>	the GIF file name

### Returns :

YAPI\_SUCCESS if the call succeeds.

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



Draws a single pixel at the specified position.

js	function <b>drawPixel</b> ( <b>x</b> , <b>y</b> )
nodejs	function <b>drawPixel</b> ( <b>x</b> , <b>y</b> )
php	function <b>drawPixel</b> ( <b>\$x</b> , <b>\$y</b> )
c++	int <b>drawPixel</b> ( int <b>x</b> , int <b>y</b> )
m	-(int) <b>drawPixel</b> : (int) <b>x</b> : (int) <b>y</b>
pas	function <b>drawPixel</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt): LongInt
vb	function <b>drawPixel</b> ( ) As Integer
cs	int <b>drawPixel</b> ( int <b>x</b> , int <b>y</b> )
java	int <b>drawPixel</b> ( int <b>x</b> , int <b>y</b> )
py	def <b>drawPixel</b> ( <b>x</b> , <b>y</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>drawPixel</b> <b>x y</b>

### Parameters :

**x** the distance from left of layer, in pixels

**y** the distance from top of layer, in pixels

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→drawRect()displaylayer.drawRect()****YDisplayLayer**

Draws an empty rectangle at a specified position.

```

js function drawRect( x1, y1, x2, y2)
nodejs function drawRect( x1, y1, x2, y2)
php function drawRect( $x1, $y1, $x2, $y2)
cpp int drawRect( int x1, int y1, int x2, int y2)
m -(int) drawRect : (int) x1
                        : (int) y1
                        : (int) x2
                        : (int) y2
pas function drawRect( x1: LongInt,
                        y1: LongInt,
                        x2: LongInt,
                        y2: LongInt): LongInt
vb function drawRect( ) As Integer
cs int drawRect( int x1, int y1, int x2, int y2)
java int drawRect( int x1, int y1, int x2, int y2)
py def drawRect( x1, y1, x2, y2)
cmd YDisplay target [-layer layerId] drawRect x1 y1 x2 y2

```

**Parameters :**

- x1** the distance from left of layer to the left border of the rectangle, in pixels
- y1** the distance from top of layer to the top border of the rectangle, in pixels
- x2** the distance from left of layer to the right border of the rectangle, in pixels
- y2** the distance from top of layer to the bottom border of the rectangle, in pixels

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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



**displaylayer**→**get\_display()****YDisplayLayer****displaylayer**→**display()****displaylayer.get\_display()**

Gets parent YDisplay.

js	function <b>get_display</b> ( )
nodejs	function <b>get_display</b> ( )
php	function <b>get_display</b> ( )
cpp	YDisplay* <b>get_display</b> ( )
m	-(YDisplay*) display
pas	function <b>get_display</b> ( ): TYDisplay
vb	function <b>get_display</b> ( ) As YDisplay
cs	YDisplay <b>get_display</b> ( )
java	YDisplay <b>get_display</b> ( )
py	def <b>get_display</b> ( )

Returns the parent YDisplay object of the current YDisplayLayer.

**Returns :**

an YDisplay object

**displaylayer→get\_displayHeight()**  
**displaylayer→displayHeight()**  
**displaylayer.get\_displayHeight()**

**YDisplayLayer**

Returns the display height, in pixels.

js	function <b>get_displayHeight</b> ( )
nodejs	function <b>get_displayHeight</b> ( )
php	function <b>get_displayHeight</b> ( )
cpp	int <b>get_displayHeight</b> ( )
m	-(int) displayHeight
pas	function <b>get_displayHeight</b> ( ): LongInt
vb	function <b>get_displayHeight</b> ( ) As Integer
cs	int <b>get_displayHeight</b> ( )
java	int <b>get_displayHeight</b> ( )
py	def <b>get_displayHeight</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>get_displayHeight</b>

#### Returns :

an integer corresponding to the display height, in pixels On failure, throws an exception or returns Y\_DISPLAYHEIGHT\_INVALID.

**displaylayer→get\_displayWidth()****YDisplayLayer****displaylayer→displayWidth()****displaylayer.get\_displayWidth()**

Returns the display width, in pixels.

js	function <b>get_displayWidth</b> ( )
nodejs	function <b>get_displayWidth</b> ( )
php	function <b>get_displayWidth</b> ( )
cpp	int <b>get_displayWidth</b> ( )
m	-(int) displayWidth
pas	function <b>get_displayWidth</b> ( ): LongInt
vb	function <b>get_displayWidth</b> ( ) As Integer
cs	int <b>get_displayWidth</b> ( )
java	int <b>get_displayWidth</b> ( )
py	def <b>get_displayWidth</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>get_displayWidth</b>

**Returns :**

an integer corresponding to the display width, in pixels On failure, throws an exception or returns Y\_DISPLAYWIDTH\_INVALID.

**displaylayer→get\_layerHeight()****YDisplayLayer****displaylayer→layerHeight()****displaylayer.get\_layerHeight()**

Returns the height of the layers to draw on, in pixels.

js	function <b>get_layerHeight</b> ( )
nodejs	function <b>get_layerHeight</b> ( )
php	function <b>get_layerHeight</b> ( )
cpp	int <b>get_layerHeight</b> ( )
m	-(int) layerHeight
pas	function <b>get_layerHeight</b> ( ): LongInt
vb	function <b>get_layerHeight</b> ( ) As Integer
cs	int <b>get_layerHeight</b> ( )
java	int <b>get_layerHeight</b> ( )
py	def <b>get_layerHeight</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>get_layerHeight</b>

**Returns :**

an integer corresponding to the height of the layers to draw on, in pixels

On failure, throws an exception or returns Y\_LAYERHEIGHT\_INVALID.

**displaylayer→get\_layerWidth()****YDisplayLayer****displaylayer→layerWidth()****displaylayer.get\_layerWidth()**

Returns the width of the layers to draw on, in pixels.

js	function <b>get_layerWidth</b> ( )
nodejs	function <b>get_layerWidth</b> ( )
php	function <b>get_layerWidth</b> ( )
cpp	int <b>get_layerWidth</b> ( )
m	-(int) layerWidth
pas	function <b>get_layerWidth</b> ( ): LongInt
vb	function <b>get_layerWidth</b> ( ) As Integer
cs	int <b>get_layerWidth</b> ( )
java	int <b>get_layerWidth</b> ( )
py	def <b>get_layerWidth</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>get_layerWidth</b>

**Returns :**

an integer corresponding to the width of the layers to draw on, in pixels

On failure, throws an exception or returns Y\_LAYERWIDTH\_INVALID.



**displaylayer→hide()displaylayer.hide()****YDisplayLayer**

Hides the layer.

js	function <b>hide</b> ( )
nodejs	function <b>hide</b> ( )
php	function <b>hide</b> ( )
cpp	int <b>hide</b> ( )
m	-(int) <b>hide</b>
pas	function <b>hide</b> ( ): LongInt
vb	function <b>hide</b> ( ) As Integer
cs	int <b>hide</b> ( )
java	int <b>hide</b> ( )
py	def <b>hide</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>hide</b>

The state of the layer is perserved but the layer is not displayed on the screen until the next call to `unhide()`. Hiding the layer can positively affect the drawing speed, since it postpones the rendering until all operations are completed (double-buffering).

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→lineTo()displaylayer.lineTo()****YDisplayLayer**

Draws a line from current drawing pointer position to the specified position.

js	function <b>lineTo</b> ( <b>x</b> , <b>y</b> )
nodejs	function <b>lineTo</b> ( <b>x</b> , <b>y</b> )
php	function <b>lineTo</b> ( <b>\$x</b> , <b>\$y</b> )
cpp	int <b>lineTo</b> ( int <b>x</b> , int <b>y</b> )
m	-(int) <b>lineTo</b> : (int) <b>x</b> : (int) <b>y</b>
pas	function <b>lineTo</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt): LongInt
vb	function <b>lineTo</b> ( ) As Integer
cs	int <b>lineTo</b> ( int <b>x</b> , int <b>y</b> )
java	int <b>lineTo</b> ( int <b>x</b> , int <b>y</b> )
py	def <b>lineTo</b> ( <b>x</b> , <b>y</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>lineTo</b> <b>x y</b>

The specified destination pixel is included in the line. The pointer position is then moved to the end point of the line.

**Parameters :**

- x** the distance from left of layer to the end point of the line, in pixels
- y** the distance from top of layer to the end point of the line, in pixels

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

Moves the drawing pointer of this layer to the specified position.

js	function <b>moveTo</b> ( <b>x</b> , <b>y</b> )
node.js	function <b>moveTo</b> ( <b>x</b> , <b>y</b> )
php	function <b>moveTo</b> ( <b>\$x</b> , <b>\$y</b> )
cpp	int <b>moveTo</b> ( int <b>x</b> , int <b>y</b> )
m	-(int) <b>moveTo</b> : (int) <b>x</b> : (int) <b>y</b>
pas	function <b>moveTo</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt): LongInt
vb	function <b>moveTo</b> ( ) As Integer
cs	int <b>moveTo</b> ( int <b>x</b> , int <b>y</b> )
java	int <b>moveTo</b> ( int <b>x</b> , int <b>y</b> )
py	def <b>moveTo</b> ( <b>x</b> , <b>y</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>moveTo</b> <b>x y</b>

### Parameters :

**x** the distance from left of layer, in pixels

**y** the distance from top of layer, in pixels

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→reset()displaylayer.reset()****YDisplayLayer**

Reverts the layer to its initial state (fully transparent, default settings).

js	function <b>reset</b> ( )
nodejs	function <b>reset</b> ( )
php	function <b>reset</b> ( )
cpp	int <b>reset</b> ( )
m	-(int) <b>reset</b>
pas	function <b>reset</b> ( ): LongInt
vb	function <b>reset</b> ( ) As Integer
cs	int <b>reset</b> ( )
java	int <b>reset</b> ( )
py	def <b>reset</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>reset</b>

Reinitializes the drawing pointer to the upper left position, and selects the most visible pen color. If you only want to erase the layer content, use the method `clear( )` instead.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## displaylayer→selectColorPen() displaylayer.selectColorPen()

## YDisplayLayer

Selects the pen color for all subsequent drawing functions, including text drawing.

js	function <b>selectColorPen</b> ( <b>color</b> )
nodejs	function <b>selectColorPen</b> ( <b>color</b> )
php	function <b>selectColorPen</b> ( <b>\$color</b> )
cpp	int <b>selectColorPen</b> ( int <b>color</b> )
m	-(int) <b>selectColorPen</b> : (int) <b>color</b>
pas	function <b>selectColorPen</b> ( <b>color</b> : LongInt): LongInt
vb	function <b>selectColorPen</b> ( ) As Integer
cs	int <b>selectColorPen</b> ( int <b>color</b> )
java	int <b>selectColorPen</b> ( int <b>color</b> )
py	def <b>selectColorPen</b> ( <b>color</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>selectColorPen color</b>

The pen color is provided as an RGB value. For grayscale or monochrome displays, the value is automatically converted to the proper range.

### Parameters :

**color** the desired pen color, as a 24-bit RGB value

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

## displaylayer→selectEraser() displaylayer.selectEraser()

YDisplayLayer

Selects an eraser instead of a pen for all subsequent drawing functions, except for text drawing and bitmap copy functions.

js	function <b>selectEraser</b> ( )
nodejs	function <b>selectEraser</b> ( )
php	function <b>selectEraser</b> ( )
cpp	int <b>selectEraser</b> ( )
m	-(int) <b>selectEraser</b>
pas	function <b>selectEraser</b> ( ): LongInt
vb	function <b>selectEraser</b> ( ) As Integer
cs	int <b>selectEraser</b> ( )
java	int <b>selectEraser</b> ( )
py	def <b>selectEraser</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>selectEraser</b>

Any point drawn using the eraser becomes transparent (as when the layer is empty), showing the other layers beneath it.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→selectFont()displaylayer.selectFont()****YDisplayLayer**

Selects a font to use for the next text drawing functions, by providing the name of the font file.

js	function <b>selectFont</b> ( <b>fontname</b> )
nodejs	function <b>selectFont</b> ( <b>fontname</b> )
php	function <b>selectFont</b> ( <b>\$fontname</b> )
cpp	int <b>selectFont</b> ( string <b>fontname</b> )
m	-(int) <b>selectFont</b> : (NSString*) <b>fontname</b>
pas	function <b>selectFont</b> ( <b>fontname</b> : string): LongInt
vb	function <b>selectFont</b> ( ) As Integer
cs	int <b>selectFont</b> ( string <b>fontname</b> )
java	int <b>selectFont</b> ( String <b>fontname</b> )
py	def <b>selectFont</b> ( <b>fontname</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>selectFont</b> <b>fontname</b>

You can use a built-in font as well as a font file that you have previously uploaded to the device built-in memory. If you experience problems selecting a font file, check the device logs for any error message such as missing font file or bad font file format.

**Parameters :**

**fontname** the font file name

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## displaylayer→selectGrayPen() displaylayer.selectGrayPen()

YDisplayLayer

Selects the pen gray level for all subsequent drawing functions, including text drawing.

js	function <b>selectGrayPen</b> ( <b>graylevel</b> )
nodejs	function <b>selectGrayPen</b> ( <b>graylevel</b> )
php	function <b>selectGrayPen</b> ( <b>\$graylevel</b> )
cpp	int <b>selectGrayPen</b> ( int <b>graylevel</b> )
m	-(int) <b>selectGrayPen</b> : (int) <b>graylevel</b>
pas	function <b>selectGrayPen</b> ( <b>graylevel</b> : LongInt): LongInt
vb	function <b>selectGrayPen</b> ( ) As Integer
cs	int <b>selectGrayPen</b> ( int <b>graylevel</b> )
java	int <b>selectGrayPen</b> ( int <b>graylevel</b> )
py	def <b>selectGrayPen</b> ( <b>graylevel</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>selectGrayPen</b> <b>graylevel</b>

The gray level is provided as a number between 0 (black) and 255 (white, or whichever the highest color is). For monochrome displays (without gray levels), any value lower than 128 is rendered as black, and any value equal or above to 128 is non-black.

### Parameters :

**graylevel** the desired gray level, from 0 to 255

### Returns :

YAPI\_SUCCESS if the call succeeds.

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



## displaylayer→setAntialiasingMode() displaylayer.setAntialiasingMode()

YDisplayLayer

Enables or disables anti-aliasing for drawing oblique lines and circles.

js	function <b>setAntialiasingMode</b> ( <b>mode</b> )
nodejs	function <b>setAntialiasingMode</b> ( <b>mode</b> )
php	function <b>setAntialiasingMode</b> ( <b>\$mode</b> )
cpp	int <b>setAntialiasingMode</b> ( bool <b>mode</b> )
m	-(int) <b>setAntialiasingMode</b> : (bool) <b>mode</b>
pas	function <b>setAntialiasingMode</b> ( <b>mode</b> : boolean): LongInt
vb	function <b>setAntialiasingMode</b> ( ) As Integer
cs	int <b>setAntialiasingMode</b> ( bool <b>mode</b> )
java	int <b>setAntialiasingMode</b> ( boolean <b>mode</b> )
py	def <b>setAntialiasingMode</b> ( <b>mode</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>setAntialiasingMode</b> <b>mode</b>

Anti-aliasing provides a smoother aspect when looked from far enough, but it can add fuzzyness when the display is looked from very close. At the end of the day, it is your personal choice. Anti-aliasing is enabled by default on grayscale and color displays, but you can disable it if you prefer. This setting has no effect on monochrome displays.

### Parameters :

**mode** true to enable antialiasing, false to disable it.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

## displaylayer→setConsoleBackground() displaylayer.setConsoleBackground()

YDisplayLayer

Sets up the background color used by the `clearConsole` function and by the console scrolling feature.

js	function <b>setConsoleBackground</b> ( <b>bgcol</b> )
nodejs	function <b>setConsoleBackground</b> ( <b>bgcol</b> )
php	function <b>setConsoleBackground</b> ( <b>\$bgcol</b> )
cpp	int <b>setConsoleBackground</b> ( int <b>bgcol</b> )
m	-(int) <b>setConsoleBackground</b> : (int) <b>bgcol</b>
pas	function <b>setConsoleBackground</b> ( <b>bgcol</b> : LongInt): LongInt
vb	function <b>setConsoleBackground</b> ( ) As Integer
cs	int <b>setConsoleBackground</b> ( int <b>bgcol</b> )
java	int <b>setConsoleBackground</b> ( int <b>bgcol</b> )
py	def <b>setConsoleBackground</b> ( <b>bgcol</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>setConsoleBackground</b> <b>bgcol</b>

### Parameters :

**bgcol** the background gray level to use when scrolling (0 = black, 255 = white), or -1 for transparent

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

## displaylayer→setConsoleMargins() displaylayer.setConsoleMargins()

## YDisplayLayer

Sets up display margins for the `consoleOut` function.

js	function <b>setConsoleMargins</b> ( <b>x1</b> , <b>y1</b> , <b>x2</b> , <b>y2</b> )
nodejs	function <b>setConsoleMargins</b> ( <b>x1</b> , <b>y1</b> , <b>x2</b> , <b>y2</b> )
php	function <b>setConsoleMargins</b> ( <b>\$x1</b> , <b>\$y1</b> , <b>\$x2</b> , <b>\$y2</b> )
cpp	int <b>setConsoleMargins</b> ( int <b>x1</b> , int <b>y1</b> , int <b>x2</b> , int <b>y2</b> )
m	-(int) <b>setConsoleMargins</b> : (int) <b>x1</b> : (int) <b>y1</b> : (int) <b>x2</b> : (int) <b>y2</b>
pas	function <b>setConsoleMargins</b> ( <b>x1</b> : LongInt, <b>y1</b> : LongInt, <b>x2</b> : LongInt, <b>y2</b> : LongInt): LongInt
vb	function <b>setConsoleMargins</b> ( ) As Integer
cs	int <b>setConsoleMargins</b> ( int <b>x1</b> , int <b>y1</b> , int <b>x2</b> , int <b>y2</b> )
java	int <b>setConsoleMargins</b> ( int <b>x1</b> , int <b>y1</b> , int <b>x2</b> , int <b>y2</b> )
py	def <b>setConsoleMargins</b> ( <b>x1</b> , <b>y1</b> , <b>x2</b> , <b>y2</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>setConsoleMargins</b> <b>x1 y1 x2 y2</b>

### Parameters :

- x1** the distance from left of layer to the left margin, in pixels
- y1** the distance from top of layer to the top margin, in pixels
- x2** the distance from left of layer to the right margin, in pixels
- y2** the distance from top of layer to the bottom margin, in pixels

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→setConsoleWordWrap()**  
**displaylayer.setConsoleWordWrap()**

YDisplayLayer

Sets up the wrapping behaviour used by the `consoleOut` function.

js	function <b>setConsoleWordWrap</b> ( <b>wordwrap</b> )
nodejs	function <b>setConsoleWordWrap</b> ( <b>wordwrap</b> )
php	function <b>setConsoleWordWrap</b> ( <b>\$wordwrap</b> )
cpp	int <b>setConsoleWordWrap</b> ( bool <b>wordwrap</b> )
m	-(int) <b>setConsoleWordWrap</b> : (bool) <b>wordwrap</b>
pas	function <b>setConsoleWordWrap</b> ( <b>wordwrap</b> : boolean): LongInt
vb	function <b>setConsoleWordWrap</b> ( ) As Integer
cs	int <b>setConsoleWordWrap</b> ( bool <b>wordwrap</b> )
java	int <b>setConsoleWordWrap</b> ( boolean <b>wordwrap</b> )
py	def <b>setConsoleWordWrap</b> ( <b>wordwrap</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>setConsoleWordWrap</b> <b>wordwrap</b>

**Parameters :**

**wordwrap** `true` to wrap only between words, `false` to wrap on the last column anyway.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→setLayerPosition()**  
**displaylayer.setLayerPosition()**

## YDisplayLayer

Sets the position of the layer relative to the display upper left corner.

js	function <b>setLayerPosition</b> ( <b>x</b> , <b>y</b> , <b>scrollTime</b> )
nodejs	function <b>setLayerPosition</b> ( <b>x</b> , <b>y</b> , <b>scrollTime</b> )
php	function <b>setLayerPosition</b> ( <b>\$x</b> , <b>\$y</b> , <b>\$scrollTime</b> )
cpp	int <b>setLayerPosition</b> ( int <b>x</b> , int <b>y</b> , int <b>scrollTime</b> )
m	-(int) <b>setLayerPosition</b> : (int) <b>x</b> : (int) <b>y</b> : (int) <b>scrollTime</b>
pas	function <b>setLayerPosition</b> ( <b>x</b> : LongInt, <b>y</b> : LongInt, <b>scrollTime</b> : LongInt): LongInt
vb	function <b>setLayerPosition</b> ( ) As Integer
cs	int <b>setLayerPosition</b> ( int <b>x</b> , int <b>y</b> , int <b>scrollTime</b> )
java	int <b>setLayerPosition</b> ( int <b>x</b> , int <b>y</b> , int <b>scrollTime</b> )
py	def <b>setLayerPosition</b> ( <b>x</b> , <b>y</b> , <b>scrollTime</b> )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>setLayerPosition</b> <b>x y scrollTime</b>

When smooth scrolling is used, the display offset of the layer is automatically updated during the next milliseconds to animate the move of the layer.

### Parameters :

<b>x</b>	the distance from left of display to the upper left corner of the layer
<b>y</b>	the distance from top of display to the upper left corner of the layer
<b>scrollTime</b>	number of milliseconds to use for smooth scrolling, or 0 if the scrolling should be immediate.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→unhide()****displaylayer.unhide()****YDisplayLayer**

Shows the layer.

js	function <b>unhide</b> ( )
nodejs	function <b>unhide</b> ( )
php	function <b>unhide</b> ( )
cpp	int <b>unhide</b> ( )
m	-(int) <b>unhide</b>
pas	function <b>unhide</b> ( ): LongInt
vb	function <b>unhide</b> ( ) As Integer
cs	int <b>unhide</b> ( )
java	int <b>unhide</b> ( )
py	def <b>unhide</b> ( )
cmd	YDisplay <b>target</b> [-layer <b>layerId</b> ] <b>unhide</b>

Shows the layer again after a hide command.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## 3.15. External power supply control interface

Yoctopuce application programming interface allows you to control the power source to use for module functions that require high current. The module can also automatically disconnect the external power when a voltage drop is observed on the external power source (external battery running out of power).

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

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

### Global functions

#### yFindDualPower(func)

Retrieves a dual power control for a given identifier.

#### yFirstDualPower()

Starts the enumeration of dual power controls currently accessible.

### YDualPower methods

#### dualpower→describe()

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

#### dualpower→get\_advertisedValue()

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

#### dualpower→get\_errorMessage()

Returns the error message of the latest error with the power control.

#### dualpower→get\_errorType()

Returns the numerical error code of the latest error with the power control.

#### dualpower→get\_extVoltage()

Returns the measured voltage on the external power source, in millivolts.

#### dualpower→get\_friendlyName()

Returns a global identifier of the power control in the format MODULE\_NAME . FUNCTION\_NAME.

#### dualpower→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### dualpower→get\_functionId()

Returns the hardware identifier of the power control, without reference to the module.

#### dualpower→get\_hardwareId()

Returns the unique hardware identifier of the power control in the form SERIAL . FUNCTIONID.

#### dualpower→get\_logicalName()

Returns the logical name of the power control.

#### dualpower→get\_module()

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

**`dualpower→get_module_async(callback, context)`**

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

**`dualpower→get_powerControl()`**

Returns the selected power source for module functions that require lots of current.

**`dualpower→get_powerState()`**

Returns the current power source for module functions that require lots of current.

**`dualpower→get_userData()`**

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

**`dualpower→isOnline()`**

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

**`dualpower→isOnline_async(callback, context)`**

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

**`dualpower→load(msValidity)`**

Preloads the power control cache with a specified validity duration.

**`dualpower→load_async(msValidity, callback, context)`**

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

**`dualpower→nextDualPower()`**

Continues the enumeration of dual power controls started using `yFirstDualPower()`.

**`dualpower→registerValueCallback(callback)`**

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

**`dualpower→set_logicalName(newval)`**

Changes the logical name of the power control.

**`dualpower→set_powerControl(newval)`**

Changes the selected power source for module functions that require lots of current.

**`dualpower→set_userData(data)`**

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

**`dualpower→wait_async(callback, context)`**

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



**YDualPower.FindDualPower()****YDualPower****yFindDualPower()** **YDualPower.FindDualPower()**

Retrieves a dual power control for a given identifier.

js	function <b>yFindDualPower</b> ( <b>func</b> )
nodejs	function <b>FindDualPower</b> ( <b>func</b> )
php	function <b>yFindDualPower</b> ( <b>\$func</b> )
cpp	YDualPower* <b>yFindDualPower</b> ( const string& <b>func</b> )
m	YDualPower* <b>yFindDualPower</b> ( NSString* <b>func</b> )
pas	function <b>yFindDualPower</b> ( <b>func</b> : string): TYDualPower
vb	function <b>yFindDualPower</b> ( ByVal <b>func</b> As String) As YDualPower
cs	YDualPower <b>FindDualPower</b> ( string <b>func</b> )
java	YDualPower <b>FindDualPower</b> ( String <b>func</b> )
py	def <b>FindDualPower</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the power control is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YDualPower.isOnline()` to test if the power control is indeed online at a given time. In case of ambiguity when looking for a dual power control 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 power control

**Returns :**

a `YDualPower` object allowing you to drive the power control.

**YDualPower.FirstDualPower()****YDualPower****yFirstDualPower()YDualPower.FirstDualPower()**

Starts the enumeration of dual power controls currently accessible.

js	function <b>yFirstDualPower</b> ( )
nodejs	function <b>FirstDualPower</b> ( )
php	function <b>yFirstDualPower</b> ( )
cpp	YDualPower* <b>yFirstDualPower</b> ( )
m	YDualPower* <b>yFirstDualPower</b> ( )
pas	function <b>yFirstDualPower</b> ( ): TYDualPower
vb	function <b>yFirstDualPower</b> ( ) As YDualPower
cs	YDualPower <b>FirstDualPower</b> ( )
java	YDualPower <b>FirstDualPower</b> ( )
py	def <b>FirstDualPower</b> ( )

Use the method `YDualPower.nextDualPower( )` to iterate on next dual power controls.

**Returns :**

a pointer to a `YDualPower` object, corresponding to the first dual power control currently online, or a `null` pointer if there are none.

**dualpower→describe()****dualpower.describe()****YDualPower**

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

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

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(BadCustomName.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 power control (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**dualpower→get\_advertisedValue()****YDualPower****dualpower→advertisedValue()****dualpower.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the power control (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**dualpower→get\_errorMessage()****YDualPower****dualpower→errorMessage()****dualpower.get\_errorMessage()**

Returns the error message of the latest error with the power control.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the power control object

**dualpower**→**get\_errorType()****YDualPower****dualpower**→**errorType()****dualpower.get\_errorType()**

Returns the numerical error code of the latest error with the power control.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the power control object

**dualpower→get\_extVoltage()****YDualPower****dualpower→extVoltage()dualpower.get\_extVoltage()**

Returns the measured voltage on the external power source, in millivolts.

js	function <b>get_extVoltage</b> ( )
nodejs	function <b>get_extVoltage</b> ( )
php	function <b>get_extVoltage</b> ( )
cpp	int <b>get_extVoltage</b> ( )
m	-(int) extVoltage
pas	function <b>get_extVoltage</b> ( ): LongInt
vb	function <b>get_extVoltage</b> ( ) As Integer
cs	int <b>get_extVoltage</b> ( )
java	int <b>get_extVoltage</b> ( )
py	def <b>get_extVoltage</b> ( )
cmd	YDualPower <b>target</b> <b>get_extVoltage</b>

**Returns :**

an integer corresponding to the measured voltage on the external power source, in millivolts

On failure, throws an exception or returns Y\_EXTVOLTAGE\_INVALID.

**dualpower**→**get\_friendlyName()****YDualPower****dualpower**→**friendlyName()****dualpower.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

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

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



**dualpower→get\_functionDescriptor()**  
**dualpower→functionDescriptor()**  
**dualpower.get\_functionDescriptor()**

**YDualPower**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**dualpower**→**get\_functionId()****YDualPower****dualpower**→**functionId()****dualpower.get\_functionId()**

Returns the hardware identifier of the power control, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the power control (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**dualpower**→**get\_hardwareId()****YDualPower****dualpower**→**hardwareId()****dualpower.get\_hardwareId()**

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

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

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

**Returns :**

a string that uniquely identifies the power control (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**dualpower→get\_logicalName()****YDualPower****dualpower→logicalName()****dualpower.get\_logicalName()**

Returns the logical name of the power control.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YDualPower <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the power control. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**dualpower→get\_module()****YDualPower****dualpower→module()dualpower.get\_module()**

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

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

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

## dualpower→get\_module\_async() dualpower→module\_async()

YDualPower

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.

**dualpower→get\_powerControl()****YDualPower****dualpower→powerControl()****dualpower.get\_powerControl()**

Returns the selected power source for module functions that require lots of current.

js	function <b>get_powerControl</b> ( )
nodejs	function <b>get_powerControl</b> ( )
php	function <b>get_powerControl</b> ( )
cpp	Y_POWERCONTROL_enum <b>get_powerControl</b> ( )
m	-(Y_POWERCONTROL_enum) powerControl
pas	function <b>get_powerControl</b> ( ): Integer
vb	function <b>get_powerControl</b> ( ) As Integer
cs	int <b>get_powerControl</b> ( )
java	int <b>get_powerControl</b> ( )
py	def <b>get_powerControl</b> ( )
cmd	YDualPower <b>target</b> <b>get_powerControl</b>

**Returns :**

a value among Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT and Y\_POWERCONTROL\_OFF corresponding to the selected power source for module functions that require lots of current

On failure, throws an exception or returns Y\_POWERCONTROL\_INVALID.

**dualpower→get\_powerState()****YDualPower****dualpower→powerState()****dualpower.get\_powerState()**

Returns the current power source for module functions that require lots of current.

js	function <b>get_powerState</b> ( )
nodejs	function <b>get_powerState</b> ( )
php	function <b>get_powerState</b> ( )
cpp	Y_POWERSTATE_enum <b>get_powerState</b> ( )
m	-(Y_POWERSTATE_enum) powerState
pas	function <b>get_powerState</b> ( ): Integer
vb	function <b>get_powerState</b> ( ) As Integer
cs	int <b>get_powerState</b> ( )
java	int <b>get_powerState</b> ( )
py	def <b>get_powerState</b> ( )
cmd	YDualPower <b>target</b> <b>get_powerState</b>

**Returns :**

a value among Y\_POWERSTATE\_OFF, Y\_POWERSTATE\_FROM\_USB and Y\_POWERSTATE\_FROM\_EXT corresponding to the current power source for module functions that require lots of current

On failure, throws an exception or returns Y\_POWERSTATE\_INVALID.



**dualpower**→**get\_userdata()****YDualPower****dualpower**→**userData()****dualpower.get\_userdata()**

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

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**dualpower**→**isOnline()****dualpower.isOnline()****YDualPower**

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

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

If there is a cached value for the power control 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 power control.

**Returns :**

`true` if the power control can be reached, and `false` otherwise

**dualpower→isOnline\_async()****YDualPower**

Checks if the power control 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 power control 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.

**dualpower→load()****dualpower.load()****YDualPower**

Preloads the power control cache with a specified validity duration.

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

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 traffic 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.

**dualpower→load\_async()****YDualPower**

Preloads the power control 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 traffic 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.

**dualpower→nextDualPower()**  
**dualpower.nextDualPower()****YDualPower**

Continues the enumeration of dual power controls started using `yFirstDualPower()`.

js	function <b>nextDualPower</b> ( )
nodejs	function <b>nextDualPower</b> ( )
php	function <b>nextDualPower</b> ( )
c++	YDualPower * <b>nextDualPower</b> ( )
m	-(YDualPower*) <b>nextDualPower</b>
pas	function <b>nextDualPower</b> ( ): TYDualPower
vb	function <b>nextDualPower</b> ( ) As YDualPower
cs	YDualPower <b>nextDualPower</b> ( )
java	YDualPower <b>nextDualPower</b> ( )
py	def <b>nextDualPower</b> ( )

**Returns :**

a pointer to a `YDualPower` object, corresponding to a dual power control currently online, or a `null` pointer if there are no more dual power controls to enumerate.

## dualpower→registerValueCallback() dualpower.registerValueCallback()

YDualPower

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YDualPowerValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YDualPowerValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYDualPowerValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**dualpower→set\_logicalName()****YDualPower****dualpower→setLogicalName()****dualpower.set\_logicalName()**

Changes the logical name of the power control.

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

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 power control.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.



**dualpower→set\_powerControl()****YDualPower****dualpower→setPowerControl()****dualpower.set\_powerControl()**

Changes the selected power source for module functions that require lots of current.

js	function <b>set_powerControl</b> ( <b>newval</b> )
nodejs	function <b>set_powerControl</b> ( <b>newval</b> )
php	function <b>set_powerControl</b> ( <b>\$newval</b> )
cpp	int <b>set_powerControl</b> ( Y_POWERCONTROL_enum <b>newval</b> )
m	-(int) <b>setPowerControl</b> : (Y_POWERCONTROL_enum) <b>newval</b>
pas	function <b>set_powerControl</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_powerControl</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_powerControl</b> ( int <b>newval</b> )
java	int <b>set_powerControl</b> ( int <b>newval</b> )
py	def <b>set_powerControl</b> ( <b>newval</b> )
cmd	YDualPower <b>target set_powerControl newval</b>

**Parameters :**

**newval** a value among Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT and Y\_POWERCONTROL\_OFF corresponding to the selected power source for module functions that require lots of current

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**dualpower**→**set\_userdata()****dualpower**→**setUserData()****dualpower.set\_userdata()**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**dualpower→wait\_async()****YDualPower**

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.

## 3.16. 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');
c++	#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()****yFindFiles()****YFiles.FindFiles()**

Retrieves a filesystem for a given identifier.

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

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()YFiles.FirstFiles()**

Starts the enumeration of filesystems currently accessible.

js	function <b>yFirstFiles</b> ( )
nodejs	function <b>FirstFiles</b> ( )
php	function <b>yFirstFiles</b> ( )
cpp	YFiles* <b>yFirstFiles</b> ( )
m	YFiles* <b>yFirstFiles</b> ( )
pas	function <b>yFirstFiles</b> ( ): TYFiles
vb	function <b>yFirstFiles</b> ( ) As YFiles
cs	YFiles <b>FirstFiles</b> ( )
java	YFiles <b>FirstFiles</b> ( )
py	def <b>FirstFiles</b> ( )

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(files.describe())****YFiles**

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

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

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(BadCustomName.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()files.download()****YFiles**

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

js	function <b>download</b> ( <b>pathname</b> )
node.js	function <b>download</b> ( <b>pathname</b> )
php	function <b>download</b> ( <b>\$pathname</b> )
cpp	string <b>download</b> ( string <b>pathname</b> )
m	-(NSData*) <b>download</b> : (NSString*) <b>pathname</b>
pas	function <b>download</b> ( <b>pathname</b> : string): TByteArray
vb	function <b>download</b> ( ) As Byte
py	def <b>download</b> ( <b>pathname</b> )
cmd	YFiles <b>target download pathname</b>

**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 download 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()files.format\_fs()****YFiles**

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

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

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()****files**→**advertisedValue()****files.get\_advertisedValue()**

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

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

**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()****files.get\_errorMessage()**

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

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the filesystem object

**files**→**get\_errorType()****files**→**errorType()****files.get\_errorType()**

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

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the filesystem object

**files**→**get\_filesCount()****YFiles****files**→**filesCount()****files.get\_filesCount()**

Returns the number of files currently loaded in the filesystem.

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

**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()****files**→**freeSpace()****files.get\_freeSpace()**

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

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

**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()****files.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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 example: `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()****files**→**functionDescriptor()****files.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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()****files.get\_functionId()**

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

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

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()****files**→**hardwareId()****files.get\_hardwareId()**

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

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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()****files.get\_list()**

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

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

**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()****files**→**logicalName()****files.get\_logicalName()**

Returns the logical name of the filesystem.

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

**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()****files.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	<code>YModule *</code> <b>get_module</b> ( )
m	-( <code>YModule*</code> ) module
pas	function <b>get_module</b> ( ): <code>TYModule</code>
vb	function <b>get_module</b> ( ) As <code>YModule</code>
cs	<code>YModule</code> <b>get_module</b> ( )
java	<code>YModule</code> <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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()****files→module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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()****files.get\_userData()**

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

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

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()****files.isOnline()****YFiles**

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

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

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()files.load()

## YFiles

Preloads the filesystem cache with a specified validity duration.

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

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 traffic 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 traffic 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()****files.nextFiles()****YFiles**

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

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

**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() files.registerValueCallback()

## YFiles

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YFilesValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YFilesValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYFilesValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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()files.remove()

## YFiles

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

js	function <b>remove</b> ( <b>pathname</b> )
nodejs	function <b>remove</b> ( <b>pathname</b> )
php	function <b>remove</b> ( <b>\$pathname</b> )
c++	int <b>remove</b> ( string <b>pathname</b> )
m	-(int) <b>remove</b> : (NSString*) <b>pathname</b>
pas	function <b>remove</b> ( <b>pathname</b> : string): LongInt
vb	function <b>remove</b> ( ) As Integer
cs	int <b>remove</b> ( string <b>pathname</b> )
java	int <b>remove</b> ( String <b>pathname</b> )
py	def <b>remove</b> ( <b>pathname</b> )
cmd	YFiles <b>target remove pathname</b>

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()****files.set\_logicalName()**

Changes the logical name of the filesystem.

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

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()****files**→**setUserData()****files.set\_userData()**

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

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

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()****files.upload()****YFiles**

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

js	function <b>upload</b> ( <b>pathname</b> , <b>content</b> )
nodejs	function <b>upload</b> ( <b>pathname</b> , <b>content</b> )
php	function <b>upload</b> ( <b>\$pathname</b> , <b>\$content</b> )
cpp	int <b>upload</b> ( string <b>pathname</b> , string <b>content</b> )
m	-(int) <b>upload</b> : (NSString*) <b>pathname</b> : (NSData*) <b>content</b>
pas	function <b>upload</b> ( <b>pathname</b> : string, <b>content</b> : TByteArray): LongInt
vb	procedure <b>upload</b> ( )
cs	int <b>upload</b> ( string <b>pathname</b> )
java	int <b>upload</b> ( String <b>pathname</b> )
py	def <b>upload</b> ( <b>pathname</b> , <b>content</b> )
cmd	YFiles <b>target upload</b> <b>pathname</b> <b>content</b>

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.

## 3.17. GenericSensor function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindGenericSensor(func)

Retrieves a generic sensor for a given identifier.

#### yFirstGenericSensor()

Starts the enumeration of generic sensors currently accessible.

### YGenericSensor methods

#### genericsensor→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### genericsensor→describe()

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

#### genericsensor→get\_advertisedValue()

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

#### genericsensor→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### genericsensor→get\_currentValue()

Returns the current measured value.

#### genericsensor→get\_errorMessage()

Returns the error message of the latest error with the generic sensor.

#### genericsensor→get\_errorType()

Returns the numerical error code of the latest error with the generic sensor.

#### genericsensor→get\_friendlyName()

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

#### genericsensor→get\_functionDescriptor()

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

#### genericsensor→get\_functionId()

Returns the hardware identifier of the generic sensor, without reference to the module.

#### genericsensor→get\_hardwareId()

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

**genericSensor→get\_highestValue()**

Returns the maximal value observed for the measure since the device was started.

**genericSensor→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**genericSensor→get\_logicalName()**

Returns the logical name of the generic sensor.

**genericSensor→get\_lowestValue()**

Returns the minimal value observed for the measure since the device was started.

**genericSensor→get\_module()**

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

**genericSensor→get\_module\_async(callback, context)**

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

**genericSensor→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**genericSensor→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**genericSensor→get\_resolution()**

Returns the resolution of the measured values.

**genericSensor→get\_signalRange()**

Returns the electric signal range used by the sensor.

**genericSensor→get\_signalUnit()**

Returns the measuring unit of the electrical signal used by the sensor.

**genericSensor→get\_signalValue()**

Returns the measured value of the electrical signal used by the sensor.

**genericSensor→get\_unit()**

Returns the measuring unit for the measure.

**genericSensor→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**genericSensor→get\_valueRange()**

Returns the physical value range measured by the sensor.

**genericSensor→isOnline()**

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

**genericSensor→isOnline\_async(callback, context)**

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

**genericSensor→load(msValidity)**

Preloads the generic sensor cache with a specified validity duration.

**genericSensor→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**genericSensor→load\_async(msValidity, callback, context)**

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

**genericSensor→nextGenericSensor()**

Continues the enumeration of generic sensors started using yFirstGenericSensor().

**genericSensor→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**genericsensor→registerValueCallback(callback)**

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

**genericsensor→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**genericsensor→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**genericsensor→set\_logicalName(newval)**

Changes the logical name of the generic sensor.

**genericsensor→set\_lowestValue(newval)**

Changes the recorded minimal value observed.

**genericsensor→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**genericsensor→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**genericsensor→set\_signalRange(newval)**

Changes the electric signal range used by the sensor.

**genericsensor→set\_unit(newval)**

Changes the measuring unit for the measured value.

**genericsensor→set\_userData(data)**

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

**genericsensor→set\_valueRange(newval)**

Changes the physical value range measured by the sensor.

**genericsensor→wait\_async(callback, context)**

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

## YGenericSensor.FindGenericSensor() yFindGenericSensor() YGenericSensor.FindGenericSensor()

YGenericSensor

Retrieves a generic sensor for a given identifier.

```

js    function yFindGenericSensor( func)
nodejs function FindGenericSensor( func)
php    function yFindGenericSensor( $func)
cpp    YGenericSensor* yFindGenericSensor( const string& func)
m       YGenericSensor* yFindGenericSensor( NSString* func)
pas     function yFindGenericSensor( func: string): TGenericSensor
vb      function yFindGenericSensor( ByVal func As String) As YGenericSensor
cs      YGenericSensor FindGenericSensor( string func)
java    YGenericSensor FindGenericSensor( String func)
py      def FindGenericSensor( 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 generic sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YGenericSensor.isOnline()` to test if the generic sensor is indeed online at a given time. In case of ambiguity when looking for a generic sensor 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 generic sensor

### Returns :

a `YGenericSensor` object allowing you to drive the generic sensor.



## YGenericSensor.FirstGenericSensor() yFirstGenericSensor() YGenericSensor.FirstGenericSensor()

## YGenericSensor

Starts the enumeration of generic sensors currently accessible.

js	function <b>yFirstGenericSensor</b> ( )
nodejs	function <b>FirstGenericSensor</b> ( )
php	function <b>yFirstGenericSensor</b> ( )
cpp	YGenericSensor* <b>yFirstGenericSensor</b> ( )
m	YGenericSensor* <b>yFirstGenericSensor</b> ( )
pas	function <b>yFirstGenericSensor</b> ( ): TYGenericSensor
vb	function <b>yFirstGenericSensor</b> ( ) As YGenericSensor
cs	YGenericSensor <b>FirstGenericSensor</b> ( )
java	YGenericSensor <b>FirstGenericSensor</b> ( )
py	def <b>FirstGenericSensor</b> ( )

Use the method `YGenericSensor.nextGenericSensor( )` to iterate on next generic sensors.

### Returns :

a pointer to a `YGenericSensor` object, corresponding to the first generic sensor currently online, or a null pointer if there are none.

## genericsensor→calibrateFromPoints() genericsensor.calibrateFromPoints()

YGenericSensor

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YGenericSensor target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→describe()genericsensor.describe()****YGenericSensor**

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

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

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(BadCustomName.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 generic sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**genericsensor**→**get\_advertisedValue()**  
**genericsensor**→**advertisedValue()**  
**genericsensor.get\_advertisedValue()**

**YGenericSensor**

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

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

**Returns :**

a string corresponding to the current value of the generic sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**genericsensor→get\_currentRawValue()****YGenericSensor****genericsensor→currentRawValue()****genericsensor.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**genericsensor→get\_currentValue()**  
**genericsensor→currentValue()**  
**genericsensor.get\_currentValue()**

**YGenericSensor**

Returns the current measured value.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measured value

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**genericsensor→get\_errorMessage()**  
**genericsensor→errorMessage()**  
**genericsensor.get\_errorMessage()**

**YGenericSensor**

Returns the error message of the latest error with the generic sensor.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the generic sensor object

**genericsensor→get\_errorType()****YGenericSensor****genericsensor→errorType()****genericsensor.get\_errorType()**

Returns the numerical error code of the latest error with the generic sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the generic sensor object



**genericsensor→get\_friendlyName()****YGenericSensor****genericsensor→friendlyName()****genericsensor.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

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

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

**genericsensor→get\_functionDescriptor()**  
**genericsensor→functionDescriptor()**  
**genericsensor.get\_functionDescriptor()**

**YGenericSensor**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**genericsensor→get\_functionId()**  
**genericsensor→functionId()**  
**genericsensor.get\_functionId()**

**YGenericSensor**

Returns the hardware identifier of the generic sensor, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the generic sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**genericsensor→get\_hardwareId()**  
**genericsensor→hardwareId()**  
**genericsensor.get\_hardwareId()**

**YGenericSensor**

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

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

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

**Returns :**

a string that uniquely identifies the generic sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**genericsensor→get\_highestValue()****YGenericSensor****genericsensor→highestValue()****genericsensor.get\_highestValue()**

Returns the maximal value observed for the measure since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the measure since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**genericsensor→get\_logFrequency()**  
**genericsensor→logFrequency()**  
**genericsensor.get\_logFrequency()**

**YGenericSensor**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**genericsensor→get\_logicalName()**  
**genericsensor→logicalName()**  
**genericsensor.get\_logicalName()**

**YGenericSensor**

Returns the logical name of the generic sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the generic sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**genericsensor→get\_lowestValue()**  
**genericsensor→lowestValue()**  
**genericsensor.get\_lowestValue()**

**YGenericSensor**

Returns the minimal value observed for the measure since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the measure since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.



**genericsensor→get\_module()****YGenericSensor****genericsensor→module()****genericsensor.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**genericsensor→get\_module\_async()**  
**genericsensor→module\_async()****YGenericSensor**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**genericsensor→get\_recordedData()****YGenericSensor****genericsensor→recordedData()****genericsensor.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YGenericSensor <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**genericsensor→get\_reportFrequency()****YGenericSensor****genericsensor→reportFrequency()****genericsensor.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**genericsensor→get\_resolution()****YGenericSensor****genericsensor→resolution()****genericsensor.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**genericsensor→get\_signalRange()**  
**genericsensor→signalRange()**  
**genericsensor.get\_signalRange()**

**YGenericSensor**

Returns the electric signal range used by the sensor.

js	function <b>get_signalRange</b> ( )
nodejs	function <b>get_signalRange</b> ( )
php	function <b>get_signalRange</b> ( )
cpp	string <b>get_signalRange</b> ( )
m	-(NSString*) signalRange
pas	function <b>get_signalRange</b> ( ): string
vb	function <b>get_signalRange</b> ( ) As String
cs	string <b>get_signalRange</b> ( )
java	String <b>get_signalRange</b> ( )
py	def <b>get_signalRange</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_signalRange</b>

**Returns :**

a string corresponding to the electric signal range used by the sensor

On failure, throws an exception or returns Y\_SIGNALRANGE\_INVALID.

**genericsensor→get\_signalUnit()****YGenericSensor****genericsensor→signalUnit()****genericsensor.get\_signalUnit()**

Returns the measuring unit of the electrical signal used by the sensor.

js	function <b>get_signalUnit</b> ( )
nodejs	function <b>get_signalUnit</b> ( )
php	function <b>get_signalUnit</b> ( )
cpp	string <b>get_signalUnit</b> ( )
m	-(NSString*) signalUnit
pas	function <b>get_signalUnit</b> ( ): string
vb	function <b>get_signalUnit</b> ( ) As String
cs	string <b>get_signalUnit</b> ( )
java	String <b>get_signalUnit</b> ( )
py	def <b>get_signalUnit</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_signalUnit</b>

**Returns :**

a string corresponding to the measuring unit of the electrical signal used by the sensor

On failure, throws an exception or returns Y\_SIGNALUNIT\_INVALID.

**genericsensor→get\_signalValue()****YGenericSensor****genericsensor→signalValue()****genericsensor.get\_signalValue()**

Returns the measured value of the electrical signal used by the sensor.

js	function <b>get_signalValue</b> ( )
nodejs	function <b>get_signalValue</b> ( )
php	function <b>get_signalValue</b> ( )
cpp	double <b>get_signalValue</b> ( )
m	-(double) signalValue
pas	function <b>get_signalValue</b> ( ): double
vb	function <b>get_signalValue</b> ( ) As Double
cs	double <b>get_signalValue</b> ( )
java	double <b>get_signalValue</b> ( )
py	def <b>get_signalValue</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_signalValue</b>

**Returns :**

a floating point number corresponding to the measured value of the electrical signal used by the sensor

On failure, throws an exception or returns Y\_SIGNALVALUE\_INVALID.



**genericsensor**→**get\_unit()****YGenericSensor****genericsensor**→**unit()****genericsensor.get\_unit()**

Returns the measuring unit for the measure.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the measure

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**genericsensor→get\_userData()****YGenericSensor****genericsensor→userData()****genericsensor.get\_userData()**

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

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

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.

**genericsensor→get\_valueRange()**  
**genericsensor→valueRange()**  
**genericsensor.get\_valueRange()**

**YGenericSensor**

Returns the physical value range measured by the sensor.

js	function <b>get_valueRange</b> ( )
nodejs	function <b>get_valueRange</b> ( )
php	function <b>get_valueRange</b> ( )
cpp	string <b>get_valueRange</b> ( )
m	-(NSString*) valueRange
pas	function <b>get_valueRange</b> ( ): string
vb	function <b>get_valueRange</b> ( ) As String
cs	string <b>get_valueRange</b> ( )
java	String <b>get_valueRange</b> ( )
py	def <b>get_valueRange</b> ( )
cmd	YGenericSensor <b>target</b> <b>get_valueRange</b>

**Returns :**

a string corresponding to the physical value range measured by the sensor

On failure, throws an exception or returns Y\_VALUERANGE\_INVALID.

**genericsensor**→**isOnline()****genericsensor.isOnline()****YGenericSensor**

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

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

If there is a cached value for the generic sensor 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 generic sensor.

**Returns :**

`true` if the generic sensor can be reached, and `false` otherwise

**genericsensor→isOnline\_async()****YGenericSensor**

Checks if the generic sensor 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 generic sensor 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.

**genericsensor→load()****genericsensor.load()****YGenericSensor**

Preloads the generic sensor cache with a specified validity duration.

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

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 traffic 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.

## genericsensor→loadCalibrationPoints() genericsensor.loadCalibrationPoints()

YGenericSensor

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js      function loadCalibrationPoints( rawValues, refValues)
nodejs  function loadCalibrationPoints( rawValues, refValues)
php     function loadCalibrationPoints( &$rawValues, &$refValues)
cpp     int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m       -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )

cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)

cmd     YGenericSensor target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→load\_async()****YGenericSensor**

Preloads the generic sensor 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 traffic 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.



**genericSensor→nextGenericSensor()**  
**genericSensor.nextGenericSensor()****YGenericSensor**

Continues the enumeration of generic sensors started using `yFirstGenericSensor()`.

js	function <b>nextGenericSensor</b> ( )
nodejs	function <b>nextGenericSensor</b> ( )
php	function <b>nextGenericSensor</b> ( )
cpp	YGenericSensor * <b>nextGenericSensor</b> ( )
m	-(YGenericSensor*) <b>nextGenericSensor</b>
pas	function <b>nextGenericSensor</b> ( ): TYGenericSensor
vb	function <b>nextGenericSensor</b> ( ) As YGenericSensor
cs	YGenericSensor <b>nextGenericSensor</b> ( )
java	YGenericSensor <b>nextGenericSensor</b> ( )
py	def <b>nextGenericSensor</b> ( )

**Returns :**

a pointer to a `YGenericSensor` object, corresponding to a generic sensor currently online, or a `null` pointer if there are no more generic sensors to enumerate.

## genericsensor→registerTimedReportCallback() genericsensor.registerTimedReportCallback()

YGenericSensor

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YGenericSensorTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YGenericSensorTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYGenericSensorTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## genericsensor→registerValueCallback() genericsensor.registerValueCallback()

## YGenericSensor

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YGenericSensorValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YGenericSensorValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYGenericSensorValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**genericsensor→set\_highestValue()**  
**genericsensor→setHighestValue()**  
**genericsensor.set\_highestValue()**

YGenericSensor

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_highestValue</b> <b>newval</b>

#### Parameters :

**newval** a floating point number corresponding to the recorded maximal value observed

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→set\_logFrequency()**  
**genericsensor→setLogFrequency()**  
**genericsensor.set\_logFrequency()**

**YGenericSensor**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_logFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the datalogger recording frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→set\_logicalName()**  
**genericsensor→setLogicalName()**  
**genericsensor.set\_logicalName()**

YGenericSensor

Changes the logical name of the generic sensor.

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

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 generic sensor.

#### Returns :

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**genericsensor→set\_lowestValue()**  
**genericsensor→setLowestValue()**  
**genericsensor.set\_lowestValue()**

**YGenericSensor**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_lowestValue</b> <b>newval</b>

#### Parameters :

**newval** a floating point number corresponding to the recorded minimal value observed

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→set\_reportFrequency()**  
**genericsensor→setReportFrequency()**  
**genericsensor.set\_reportFrequency()**

YGenericSensor

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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



**genericsensor→set\_resolution()**  
**genericsensor→setResolution()**  
**genericsensor.set\_resolution()**

**YGenericSensor**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_resolution</b> <b>newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

#### Parameters :

**newval** a floating point number corresponding to the resolution of the measured physical values

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→set\_signalRange()****YGenericSensor****genericsensor→setSignalRange()****genericsensor.set\_signalRange()**

Changes the electric signal range used by the sensor.

js	function <b>set_signalRange</b> ( <b>newval</b> )
nodejs	function <b>set_signalRange</b> ( <b>newval</b> )
php	function <b>set_signalRange</b> ( <b>\$newval</b> )
cpp	int <b>set_signalRange</b> ( const string& <b>newval</b> )
m	-(int) setSignalRange : (NSString*) <b>newval</b>
pas	function <b>set_signalRange</b> ( <b>newval</b> : string): integer
vb	function <b>set_signalRange</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_signalRange</b> ( string <b>newval</b> )
java	int <b>set_signalRange</b> ( String <b>newval</b> )
py	def <b>set_signalRange</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_signalRange</b> <b>newval</b>

**Parameters :**

**newval** a string corresponding to the electric signal range used by the sensor

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor**→**set\_unit()****YGenericSensor****genericsensor**→**setUnit()****genericsensor.set\_unit()**

Changes the measuring unit for the measured value.

js	function <b>set_unit</b> ( <b>newval</b> )
nodejs	function <b>set_unit</b> ( <b>newval</b> )
php	function <b>set_unit</b> ( <b>\$newval</b> )
cpp	int <b>set_unit</b> ( const string& <b>newval</b> )
m	-(int) setUnit : (NSString*) <b>newval</b>
pas	function <b>set_unit</b> ( <b>newval</b> : string): integer
vb	function <b>set_unit</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_unit</b> ( string <b>newval</b> )
java	int <b>set_unit</b> ( String <b>newval</b> )
py	def <b>set_unit</b> ( <b>newval</b> )
cmd	<b>YGenericSensor target set_unit newval</b>

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

**Parameters :**

**newval** a string corresponding to the measuring unit for the measured value

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→set\_userdata()**  
**genericsensor→setUserData()**  
**genericsensor.set\_userdata()**

**YGenericSensor**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**genericsensor→set\_valueRange()****YGenericSensor****genericsensor→setValueRange()****genericsensor.set\_valueRange()**

Changes the physical value range measured by the sensor.

js	function <b>set_valueRange</b> ( <b>newval</b> )
nodejs	function <b>set_valueRange</b> ( <b>newval</b> )
php	function <b>set_valueRange</b> ( <b>\$newval</b> )
cpp	int <b>set_valueRange</b> ( const string& <b>newval</b> )
m	-(int) setValueRange : (NSString*) <b>newval</b>
pas	function <b>set_valueRange</b> ( <b>newval</b> : string): integer
vb	function <b>set_valueRange</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_valueRange</b> ( string <b>newval</b> )
java	int <b>set_valueRange</b> ( String <b>newval</b> )
py	def <b>set_valueRange</b> ( <b>newval</b> )
cmd	YGenericSensor <b>target</b> <b>set_valueRange</b> <b>newval</b>

The range change may have a side effect on the display resolution, as it may be adapted automatically.

**Parameters :**

**newval** a string corresponding to the physical value range measured by the sensor

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**genericsensor→wait\_async()****YGenericSensor**

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.

## 3.18. Gyroscope function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindGyro(func)

Retrieves a gyroscope for a given identifier.

#### yFirstGyro()

Starts the enumeration of gyroscopes currently accessible.

### YGyro methods

#### gyro→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### gyro→describe()

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

#### gyro→get\_advertisedValue()

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

#### gyro→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### gyro→get\_currentValue()

Returns the current value of the angular velocity.

#### gyro→get\_errorMessage()

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

#### gyro→get\_errorType()

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

#### gyro→get\_friendlyName()

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

#### gyro→get\_functionDescriptor()

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

#### gyro→get\_functionId()

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

#### gyro→get\_hardwareId()

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

**gyro→get\_heading()**

Returns the estimated heading angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_highestValue()**

Returns the maximal value observed for the angular velocity since the device was started.

**gyro→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**gyro→get\_logicalName()**

Returns the logical name of the gyroscope.

**gyro→get\_lowestValue()**

Returns the minimal value observed for the angular velocity since the device was started.

**gyro→get\_module()**

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

**gyro→get\_module\_async(callback, context)**

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

**gyro→get\_pitch()**

Returns the estimated pitch angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_quaternionW()**

Returns the w component (real part) of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_quaternionX()**

Returns the x component of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_quaternionY()**

Returns the y component of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_quaternionZ()**

Returns the z component of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**gyro→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**gyro→get\_resolution()**

Returns the resolution of the measured values.

**gyro→get\_roll()**

Returns the estimated roll angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

**gyro→get\_unit()**

Returns the measuring unit for the angular velocity.

**gyro→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**gyro→get\_xValue()**

Returns the angular velocity around the X axis of the device, as a floating point number.

**gyro→get\_yValue()**



Returns the angular velocity around the Y axis of the device, as a floating point number.

**gyro→get\_zValue()**

Returns the angular velocity around the Z axis of the device, as a floating point number.

**gyro→isOnline()**

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

**gyro→isOnline\_async(callback, context)**

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

**gyro→load(msValidity)**

Preloads the gyroscope cache with a specified validity duration.

**gyro→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

**gyro→load\_async(msValidity, callback, context)**

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

**gyro→nextGyro()**

Continues the enumeration of gyroscopes started using `yFirstGyro()`.

**gyro→registerAnglesCallback(callback)**

Registers a callback function that will be invoked each time that the estimated device orientation has changed.

**gyro→registerQuaternionCallback(callback)**

Registers a callback function that will be invoked each time that the estimated device orientation has changed.

**gyro→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**gyro→registerValueCallback(callback)**

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

**gyro→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**gyro→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**gyro→set\_logicalName(newval)**

Changes the logical name of the gyroscope.

**gyro→set\_lowestValue(newval)**

Changes the recorded minimal value observed.

**gyro→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**gyro→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**gyro→set\_userData(data)**

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

**gyro→wait\_async(callback, context)**

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

## YGyro.FindGyro() yFindGyro()YGyro.FindGyro()

YGyro

Retrieves a gyroscope for a given identifier.

js	function <b>yFindGyro</b> ( <b>func</b> )
nodejs	function <b>FindGyro</b> ( <b>func</b> )
php	function <b>yFindGyro</b> ( <b>\$func</b> )
cpp	YGyro* <b>yFindGyro</b> ( string <b>func</b> )
m	+(YGyro*) <b>yFindGyro</b> : (NSString*) <b>func</b>
pas	function <b>yFindGyro</b> ( <b>func</b> : string): TYGyro
vb	function <b>yFindGyro</b> ( ByVal <b>func</b> As String) As YGyro
cs	YGyro <b>FindGyro</b> ( string <b>func</b> )
java	YGyro <b>FindGyro</b> ( String <b>func</b> )
py	def <b>FindGyro</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the gyroscope is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YGyro.isOnline()` to test if the gyroscope is indeed online at a given time. In case of ambiguity when looking for a gyroscope 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 gyroscope

### Returns :

a YGyro object allowing you to drive the gyroscope.

## YGyro.FirstGyro() yFirstGyro()YGyro.FirstGyro()

## YGyro

Starts the enumeration of gyroscopes currently accessible.

js	function <b>yFirstGyro</b> ( )
nodejs	function <b>FirstGyro</b> ( )
php	function <b>yFirstGyro</b> ( )
cpp	YGyro* <b>yFirstGyro</b> ( )
m	YGyro* <b>yFirstGyro</b> ( )
pas	function <b>yFirstGyro</b> ( ): TYGyro
vb	function <b>yFirstGyro</b> ( ) As YGyro
cs	YGyro <b>FirstGyro</b> ( )
java	YGyro <b>FirstGyro</b> ( )
py	def <b>FirstGyro</b> ( )

Use the method `YGyro.nextGyro( )` to iterate on next gyroscopes.

### Returns :

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

## gyro→calibrateFromPoints() gyro.calibrateFromPoints()

YGyro

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                 refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                             ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YGyro target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**gyro→describe()****gyro.describe()****YGyro**

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

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

More precisely, `TYPE` is the type of the function, `NAME` is 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(BadCustomName.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 gyroscope (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**gyro**→**get\_advertisedValue()****YGyro****gyro**→**advertisedValue()****gyro.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the gyroscope (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**gyro→get\_currentRawValue()****YGyro****gyro→currentRawValue()gyro.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YGyro <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**gyro**→**get\_currentValue()****YGyro****gyro**→**currentValue()****gyro.get\_currentValue()**

Returns the current value of the angular velocity.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YGyro <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the angular velocity

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.



**gyro→get\_errorMessage()****YGyro****gyro→errorMessage()gyro.get\_errorMessage()**

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

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the gyroscope object

**gyro**→**get\_errorType()****YGyro****gyro**→**errorType()****gyro.get\_errorType()**

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

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the gyroscope object

**gyro→get\_friendlyName()****YGyro****gyro→friendlyName()gyro.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the gyroscope using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**gyro→get\_functionDescriptor()****gyro→functionDescriptor()****gyro.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.

**gyro**→**get\_functionId()****YGyro****gyro**→**functionId()****gyro.get\_functionId()**

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

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

For example `relay1`

**Returns :**

a string that identifies the gyroscope (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**gyro**→**get\_hardwareId()****YGyro****gyro**→**hardwareId()****gyro.get\_hardwareId()**

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

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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

**Returns :**

a string that uniquely identifies the gyroscope (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**gyro**→**get\_heading()****YGyro****gyro**→**heading()****gyro.get\_heading()**

Returns the estimated heading angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_heading</b> ( )
nodejs	function <b>get_heading</b> ( )
php	function <b>get_heading</b> ( )
cpp	double <b>get_heading</b> ( )
m	-(double) heading
pas	function <b>get_heading</b> ( ): double
vb	function <b>get_heading</b> ( ) As Double
cs	double <b>get_heading</b> ( )
java	double <b>get_heading</b> ( )
py	def <b>get_heading</b> ( )

The axis corresponding to the heading can be mapped to any of the device X, Y or Z physical directions using methods of the class `YRefFrame`.

**Returns :**

a floating-point number corresponding to heading in degrees, between 0 and 360.

**gyro**→**get\_highestValue()****YGyro****gyro**→**highestValue()****gyro.get\_highestValue()**

Returns the maximal value observed for the angular velocity since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YGyro <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the angular velocity since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.



**gyro→get\_logFrequency()****YGyro****gyro→logFrequency()gyro.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YGyro <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**gyro**→**get\_logicalName()****YGyro****gyro**→**logicalName()****gyro.get\_logicalName()**

Returns the logical name of the gyroscope.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YGyro <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the gyroscope. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**gyro**→**get\_lowestValue()****YGyro****gyro**→**lowestValue()****gyro.get\_lowestValue()**

Returns the minimal value observed for the angular velocity since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YGyro <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the angular velocity since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**gyro**→**get\_module()****YGyro****gyro**→**module()****gyro.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**gyro→get\_module\_async()**  
**gyro→module\_async()****YGyro**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**gyro**→**get\_pitch()****YGyro****gyro**→**pitch()****gyro.get\_pitch()**

Returns the estimated pitch angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_pitch</b> ( )
nodejs	function <b>get_pitch</b> ( )
php	function <b>get_pitch</b> ( )
cpp	double <b>get_pitch</b> ( )
m	-(double) pitch
pas	function <b>get_pitch</b> ( ): double
vb	function <b>get_pitch</b> ( ) As Double
cs	double <b>get_pitch</b> ( )
java	double <b>get_pitch</b> ( )
py	def <b>get_pitch</b> ( )

The axis corresponding to the pitch angle can be mapped to any of the device X, Y or Z physical directions using methods of the class `YRefFrame`.

**Returns :**

a floating-point number corresponding to pitch angle in degrees, between -90 and +90.

**gyro**→**get\_quaternionW()****YGyro****gyro**→**quaternionW()****gyro.get\_quaternionW()**

Returns the w component (real part) of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_quaternionW</b> ( )
nodejs	function <b>get_quaternionW</b> ( )
php	function <b>get_quaternionW</b> ( )
cpp	double <b>get_quaternionW</b> ( )
m	-(double) quaternionW
pas	function <b>get_quaternionW</b> ( ): double
vb	function <b>get_quaternionW</b> ( ) As Double
cs	double <b>get_quaternionW</b> ( )
java	double <b>get_quaternionW</b> ( )
py	def <b>get_quaternionW</b> ( )

**Returns :**

a floating-point number corresponding to the w component of the quaternion.

**gyro**→**get\_quaternionX()****YGyro****gyro**→**quaternionX()****gyro.get\_quaternionX()**

Returns the  $x$  component of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_quaternionX</b> ( )
nodejs	function <b>get_quaternionX</b> ( )
php	function <b>get_quaternionX</b> ( )
cpp	double <b>get_quaternionX</b> ( )
m	-(double) quaternionX
pas	function <b>get_quaternionX</b> ( ): double
vb	function <b>get_quaternionX</b> ( ) As Double
cs	double <b>get_quaternionX</b> ( )
java	double <b>get_quaternionX</b> ( )
py	def <b>get_quaternionX</b> ( )

The  $x$  component is mostly correlated with rotations on the roll axis.

**Returns :**

a floating-point number corresponding to the  $x$  component of the quaternion.



**gyro**→**get\_quaternionY()****YGyro****gyro**→**quaternionY()****gyro.get\_quaternionY()**

Returns the  $y$  component of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_quaternionY</b> ( )
nodejs	function <b>get_quaternionY</b> ( )
php	function <b>get_quaternionY</b> ( )
cpp	double <b>get_quaternionY</b> ( )
m	-(double) quaternionY
pas	function <b>get_quaternionY</b> ( ): double
vb	function <b>get_quaternionY</b> ( ) As Double
cs	double <b>get_quaternionY</b> ( )
java	double <b>get_quaternionY</b> ( )
py	def <b>get_quaternionY</b> ( )

The  $y$  component is mostly correlated with rotations on the pitch axis.

**Returns :**

a floating-point number corresponding to the  $y$  component of the quaternion.

**gyro**→**get\_quaternionZ()****YGyro****gyro**→**quaternionZ()****gyro.get\_quaternionZ()**

Returns the  $x$  component of the quaternion describing the device estimated orientation, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_quaternionZ</b> ( )
nodejs	function <b>get_quaternionZ</b> ( )
php	function <b>get_quaternionZ</b> ( )
cpp	double <b>get_quaternionZ</b> ( )
m	-(double) quaternionZ
pas	function <b>get_quaternionZ</b> ( ): double
vb	function <b>get_quaternionZ</b> ( ) As Double
cs	double <b>get_quaternionZ</b> ( )
java	double <b>get_quaternionZ</b> ( )
py	def <b>get_quaternionZ</b> ( )

The  $x$  component is mostly correlated with changes of heading.

**Returns :**

a floating-point number corresponding to the  $z$  component of the quaternion.

**gyro→get\_recordedData()****YGyro****gyro→recordedData()gyro.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YGyro <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**gyro→get\_reportFrequency()****YGyro****gyro→reportFrequency()gyro.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YGyro <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**gyro→get\_resolution()****YGyro****gyro→resolution()****gyro.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YGyro <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**gyro**→**get\_roll()****YGyro****gyro**→**roll()****gyro.get\_roll()**

Returns the estimated roll angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.

js	function <b>get_roll</b> ( )
nodejs	function <b>get_roll</b> ( )
php	function <b>get_roll</b> ( )
cpp	double <b>get_roll</b> ( )
m	-(double) roll
pas	function <b>get_roll</b> ( ): double
vb	function <b>get_roll</b> ( ) As Double
cs	double <b>get_roll</b> ( )
java	double <b>get_roll</b> ( )
py	def <b>get_roll</b> ( )

The axis corresponding to the roll angle can be mapped to any of the device X, Y or Z physical directions using methods of the class `YRefFrame`.

**Returns :**

a floating-point number corresponding to roll angle in degrees, between -180 and +180.

**gyro**→**get\_unit()****YGyro****gyro**→**unit()****gyro.get\_unit()**

Returns the measuring unit for the angular velocity.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YGyro <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the angular velocity

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**gyro**→**get\_userData()****gyro**→**userData()****gyro.get\_userData()**

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

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

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.



**gyro→get\_xValue()****YGyro****gyro→xValue()****gyro.get\_xValue()**

Returns the angular velocity around the X axis of the device, as a floating point number.

js	function <b>get_xValue</b> ( )
nodejs	function <b>get_xValue</b> ( )
php	function <b>get_xValue</b> ( )
cpp	double <b>get_xValue</b> ( )
m	-(double) xValue
pas	function <b>get_xValue</b> ( ): double
vb	function <b>get_xValue</b> ( ) As Double
cs	double <b>get_xValue</b> ( )
java	double <b>get_xValue</b> ( )
py	def <b>get_xValue</b> ( )

**Returns :**

a floating point number corresponding to the angular velocity around the X axis of the device, as a floating point number

On failure, throws an exception or returns Y\_XVALUE\_INVALID.

**gyro**→**get\_yValue()****YGyro****gyro**→**yValue()****gyro.get\_yValue()**

Returns the angular velocity around the Y axis of the device, as a floating point number.

js	function <b>get_yValue</b> ( )
nodejs	function <b>get_yValue</b> ( )
php	function <b>get_yValue</b> ( )
cpp	double <b>get_yValue</b> ( )
m	-(double) yValue
pas	function <b>get_yValue</b> ( ): double
vb	function <b>get_yValue</b> ( ) As Double
cs	double <b>get_yValue</b> ( )
java	double <b>get_yValue</b> ( )
py	def <b>get_yValue</b> ( )

**Returns :**

a floating point number corresponding to the angular velocity around the Y axis of the device, as a floating point number

On failure, throws an exception or returns Y\_YVALUE\_INVALID.

**gyro**→**get\_zValue()****YGyro****gyro**→**zValue()****gyro.get\_zValue()**

Returns the angular velocity around the Z axis of the device, as a floating point number.

js	function <b>get_zValue</b> ( )
nodejs	function <b>get_zValue</b> ( )
php	function <b>get_zValue</b> ( )
cpp	double <b>get_zValue</b> ( )
m	-(double) zValue
pas	function <b>get_zValue</b> ( ): double
vb	function <b>get_zValue</b> ( ) As Double
cs	double <b>get_zValue</b> ( )
java	double <b>get_zValue</b> ( )
py	def <b>get_zValue</b> ( )

**Returns :**

a floating point number corresponding to the angular velocity around the Z axis of the device, as a floating point number

On failure, throws an exception or returns Y\_ZVALUE\_INVALID.

**gyro→isOnline()****gyro.isOnline()****YGyro**

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

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

If there is a cached value for the gyroscope 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 gyroscope.

**Returns :**

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

**gyro→isOnline\_async()****YGyro**

Checks if the gyroscope 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 gyroscope 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.

**gyro→load()****gyro.load()****YGyro**

Preloads the gyroscope cache with a specified validity duration.

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

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 traffic 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.

## gyro→loadCalibrationPoints() gyro.loadCalibrationPoints()

YGyro

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js      function loadCalibrationPoints( rawValues, refValues)
nodejs  function loadCalibrationPoints( rawValues, refValues)
php     function loadCalibrationPoints( &$rawValues, &$refValues)
cpp     int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m       -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )
cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)
cmd     YGyro target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**gyro→load\_async()****YGyro**

Preloads the gyroscope 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 traffic 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.



**gyro→nextGyro()gyro.nextGyro()****YGyro**

Continues the enumeration of gyroscopes started using `yFirstGyro()`.

js	function <b>nextGyro</b> ( )
nodejs	function <b>nextGyro</b> ( )
php	function <b>nextGyro</b> ( )
cpp	YGyro * <b>nextGyro</b> ( )
m	-(YGyro*) <b>nextGyro</b>
pas	function <b>nextGyro</b> ( ): TYGyro
vb	function <b>nextGyro</b> ( ) As YGyro
cs	YGyro <b>nextGyro</b> ( )
java	YGyro <b>nextGyro</b> ( )
py	def <b>nextGyro</b> ( )

**Returns :**

a pointer to a `YGyro` object, corresponding to a gyroscope currently online, or a `null` pointer if there are no more gyroscopes to enumerate.

## **gyro→registerAnglesCallback()** **gyro.registerAnglesCallback()**

**YGyro**

Registers a callback function that will be invoked each time that the estimated device orientation has changed.

js	function <b>registerAnglesCallback</b> ( <b>callback</b> )
nodejs	function <b>registerAnglesCallback</b> ( <b>callback</b> )
php	function <b>registerAnglesCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerAnglesCallback</b> ( YAnglesCallback <b>callback</b> )
m	-(int) <b>registerAnglesCallback</b> : (YAnglesCallback) <b>callback</b>
pas	function <b>registerAnglesCallback</b> ( <b>callback</b> : TYAnglesCallback): LongInt
vb	function <b>registerAnglesCallback</b> ( ) As Integer
cs	int <b>registerAnglesCallback</b> ( YAnglesCallback <b>callback</b> )
java	int <b>registerAnglesCallback</b> ( YAnglesCallback <b>callback</b> )
py	def <b>registerAnglesCallback</b> ( <b>callback</b> )

The call frequency is typically around 95Hz during a move. 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 invoke, or a null pointer. The callback function should take four arguments: the YGyro object of the turning device, and the floating point values of the three angles roll, pitch and heading in degrees (as floating-point numbers).

## gyro→registerQuaternionCallback() gyro.registerQuaternionCallback()

## YGyro

Registers a callback function that will be invoked each time that the estimated device orientation has changed.

js	function <b>registerQuaternionCallback</b> ( <b>callback</b> )
nodejs	function <b>registerQuaternionCallback</b> ( <b>callback</b> )
php	function <b>registerQuaternionCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerQuaternionCallback</b> ( YQuatCallback <b>callback</b> )
m	-(int) <b>registerQuaternionCallback</b> : (YQuatCallback) <b>callback</b>
pas	function <b>registerQuaternionCallback</b> ( <b>callback</b> : TYQuatCallback): LongInt
vb	function <b>registerQuaternionCallback</b> ( ) As Integer
cs	int <b>registerQuaternionCallback</b> ( YQuatCallback <b>callback</b> )
java	int <b>registerQuaternionCallback</b> ( YQuatCallback <b>callback</b> )
py	def <b>registerQuaternionCallback</b> ( <b>callback</b> )

The call frequency is typically around 95Hz during a move. 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 invoke, or a null pointer. The callback function should take five arguments: the YGyro object of the turning device, and the floating point values of the four components w, x, y and z (as floating-point numbers).

## gyro→registerTimedReportCallback() gyro.registerTimedReportCallback()

YGyro

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YGyroTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YGyroTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYGyroTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## gyro→registerValueCallback() gyro.registerValueCallback()

## YGyro

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YGyroValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YGyroValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYGyroValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**gyro**→**set\_highestValue()****YGyro****gyro**→**setHighestValue()****gyro.set\_highestValue()**

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YGyro <b>target set_highestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**gyro→set\_logFrequency()****YGyro****gyro→setLogFrequency()****gyro.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YGyro <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**gyro**→**set\_logicalName()****YGyro****gyro**→**setLogicalName()****gyro.set\_logicalName()**

Changes the logical name of the gyroscope.

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

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 gyroscope.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.



**gyro→set\_lowestValue()****YGyro****gyro→setLowestValue()gyro.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YGyro <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**gyro→set\_reportFrequency()****gyro→setReportFrequency()****gyro.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YGyro <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**gyro→set\_resolution()****YGyro****gyro→setResolution()gyro.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YGyro <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**gyro**→**set\_userData()****gyro**→**setUserData()****gyro.set\_userData()**

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

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

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

**gyro→wait\_async()****YGyro**

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.

## 3.19. 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');
c++	#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()YHubPort.FindHubPort()**

Retrieves a Yocto-hub port for a given identifier.

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

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()YHubPort.FirstHubPort()**

Starts the enumeration of Yocto-hub ports currently accessible.

<code>js</code>	<code>function yFirstHubPort( )</code>
<code>nodejs</code>	<code>function FirstHubPort( )</code>
<code>php</code>	<code>function yFirstHubPort( )</code>
<code>cpp</code>	<code>YHubPort* yFirstHubPort( )</code>
<code>m</code>	<code>YHubPort* yFirstHubPort( )</code>
<code>pas</code>	<code>function yFirstHubPort( ): TYHubPort</code>
<code>vb</code>	<code>function yFirstHubPort( ) As YHubPort</code>
<code>cs</code>	<code>YHubPort FirstHubPort( )</code>
<code>java</code>	<code>YHubPort FirstHubPort( )</code>
<code>py</code>	<code>def FirstHubPort( )</code>

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()****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 <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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()****hubport.get\_advertisedValue()**

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

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

**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()****hubport.get\_baudRate()**

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

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

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()****hubport.get\_enabled()**

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

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

**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()****hubport.get\_errorMessage()**

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

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

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

**Returns :**

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

---

**hubport**→**get\_errorType()****YHubPort****hubport**→**errorType()****hubport.get\_errorType()**

---

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

<code>js</code>	<code>function <b>get_errorType</b>( )</code>
<code>nodejs</code>	<code>function <b>get_errorType</b>( )</code>
<code>php</code>	<code>function <b>get_errorType</b>( )</code>
<code>cpp</code>	<code>YRETCODE <b>get_errorType</b>( )</code>
<code>pas</code>	<code>function <b>get_errorType</b>( ): YRETCODE</code>
<code>vb</code>	<code>function <b>get_errorType</b>( ) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE <b>get_errorType</b>( )</code>
<code>java</code>	<code>int <b>get_errorType</b>( )</code>
<code>py</code>	<code>def <b>get_errorType</b>( )</code>

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 occurred while using the Yocto-hub port object

**hubport**→**get\_friendlyName()****YHubPort****hubport**→**friendlyName()****hubport.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
c++	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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 example: `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()****hubport.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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()****hubport.get\_functionId()**

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

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

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()****hubport.get\_hardwareId()**

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

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

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()****hubport.get\_logicalName()**

Returns the logical name of the Yocto-hub port.

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

**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()****hubport.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	<code>YModule *</code> <b>get_module</b> ( )
m	<code>-(YModule*)</code> module
pas	function <b>get_module</b> ( ): <code>TYModule</code>
vb	function <b>get_module</b> ( ) As <code>YModule</code>
cs	<code>YModule</code> <b>get_module</b> ( )
java	<code>YModule</code> <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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 <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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()****hubport.get\_portState()**

Returns the current state of the Yocto-hub port.

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

**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()****hubport.get\_userData()**

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

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

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()hubport.isOnline()****YHubPort**

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

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

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()hubport.load()****YHubPort**

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

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

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 traffic 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 traffic 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()****hubport.nextHubPort()****YHubPort**

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

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

**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() hubport.registerValueCallback()

YHubPort

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

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

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()****hubport.set\_enabled()**

Changes the activation of the Yocto-hub port.

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

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()

hubport.set\_logicalName()

Changes the logical name of the Yocto-hub port.

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

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()hubport.set\_userdata()**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
c++	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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.

## 3.20. Humidity function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

js	<code>&lt;script type='text/javascript' src='yocto_humidity.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YHumidity = yoctolib.YHumidity;</code>
php	<code>require_once('yocto_humidity.php');</code>
c++	<code>#include "yocto_humidity.h"</code>
m	<code>#import "yocto_humidity.h"</code>
pas	<code>uses yocto_humidity;</code>
vb	<code>yocto_humidity.vb</code>
cs	<code>yocto_humidity.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YHumidity;</code>
py	<code>from yocto_humidity import *</code>

### Global functions

#### **yFindHumidity(func)**

Retrieves a humidity sensor for a given identifier.

#### **yFirstHumidity()**

Starts the enumeration of humidity sensors currently accessible.

### YHumidity methods

#### **humidity→calibrateFromPoints(rawValues, refValues)**

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### **humidity→describe()**

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

#### **humidity→get\_advertisedValue()**

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

#### **humidity→get\_currentRawValue()**

Returns the unrounded and uncalibrated raw value returned by the sensor.

#### **humidity→get\_currentValue()**

Returns the current measure for the humidity.

#### **humidity→get\_errorMessage()**

Returns the error message of the latest error with the humidity sensor.

#### **humidity→get\_errorType()**

Returns the numerical error code of the latest error with the humidity sensor.

#### **humidity→get\_friendlyName()**

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

#### **humidity→get\_functionDescriptor()**

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

#### **humidity→get\_functionId()**

Returns the hardware identifier of the humidity sensor, without reference to the module.

#### **humidity→get\_hardwareId()**

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

**humidity→get\_highestValue()**

Returns the maximal value observed for the humidity.

**humidity→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**humidity→get\_logicalName()**

Returns the logical name of the humidity sensor.

**humidity→get\_lowestValue()**

Returns the minimal value observed for the humidity.

**humidity→get\_module()**

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

**humidity→get\_module\_async(callback, context)**

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

**humidity→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**humidity→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**humidity→get\_resolution()**

Returns the resolution of the measured values.

**humidity→get\_unit()**

Returns the measuring unit for the humidity.

**humidity→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**humidity→isOnline()**

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

**humidity→isOnline\_async(callback, context)**

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

**humidity→load(msValidity)**

Preloads the humidity sensor cache with a specified validity duration.

**humidity→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**humidity→load\_async(msValidity, callback, context)**

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

**humidity→nextHumidity()**

Continues the enumeration of humidity sensors started using yFirstHumidity().

**humidity→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**humidity→registerValueCallback(callback)**

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

**humidity→set\_highestValue(newval)**

Changes the recorded maximal value observed for the humidity.

**humidity→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**humidity→set\_logicalName(newval)**

Changes the logical name of the humidity sensor.

**humidity→set\_lowestValue(newval)**

Changes the recorded minimal value observed for the humidity.

**humidity→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**humidity→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**humidity→set\_userData(data)**

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

**humidity→wait\_async(callback, context)**

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

## YHumidity.FindHumidity() yFindHumidity()YHumidity.FindHumidity()

YHumidity

Retrieves a humidity sensor for a given identifier.

js	function <b>yFindHumidity</b> ( <b>func</b> )
nodejs	function <b>FindHumidity</b> ( <b>func</b> )
php	function <b>yFindHumidity</b> ( <b>\$func</b> )
cpp	YHumidity* <b>yFindHumidity</b> ( const string& <b>func</b> )
m	YHumidity* <b>yFindHumidity</b> ( NSString* <b>func</b> )
pas	function <b>yFindHumidity</b> ( <b>func</b> : string): TYHumidity
vb	function <b>yFindHumidity</b> ( ByVal <b>func</b> As String) As YHumidity
cs	YHumidity <b>FindHumidity</b> ( string <b>func</b> )
java	YHumidity <b>FindHumidity</b> ( String <b>func</b> )
py	def <b>FindHumidity</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the humidity sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YHumidity.isOnline()` to test if the humidity sensor is indeed online at a given time. In case of ambiguity when looking for a humidity sensor 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 humidity sensor

### Returns :

a `YHumidity` object allowing you to drive the humidity sensor.

## YHumidity.FirstHumidity() yFirstHumidity()YHumidity.FirstHumidity()

## YHumidity

Starts the enumeration of humidity sensors currently accessible.

js	function <b>yFirstHumidity</b> ( )
nodejs	function <b>FirstHumidity</b> ( )
php	function <b>yFirstHumidity</b> ( )
cpp	YHumidity* <b>yFirstHumidity</b> ( )
m	YHumidity* <b>yFirstHumidity</b> ( )
pas	function <b>yFirstHumidity</b> ( ): TYHumidity
vb	function <b>yFirstHumidity</b> ( ) As YHumidity
cs	YHumidity <b>FirstHumidity</b> ( )
java	YHumidity <b>FirstHumidity</b> ( )
py	def <b>FirstHumidity</b> ( )

Use the method `YHumidity.nextHumidity( )` to iterate on next humidity sensors.

### Returns :

a pointer to a `YHumidity` object, corresponding to the first humidity sensor currently online, or a `null` pointer if there are none.

## humidity→calibrateFromPoints()

### humidity.calibrateFromPoints()

YHumidity

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YHumidity target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

#### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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



**humidity→describe()humidity.describe()****YHumidity**

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

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

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(BadCustomName.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 humidity sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**humidity**→**get\_advertisedValue()****YHumidity****humidity**→**advertisedValue()****humidity.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the humidity sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**humidity→get\_currentRawValue()****YHumidity****humidity→currentRawValue()****humidity.get\_currentRawValue()**

Returns the unrounded and uncalibrated raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YHumidity <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the unrounded and uncalibrated raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**humidity**→**get\_currentValue()****YHumidity****humidity**→**currentValue()****humidity.get\_currentValue()**

Returns the current measure for the humidity.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YHumidity <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the humidity

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

---

**humidity→get\_errorMessage()****YHumidity****humidity→errorMessage()****humidity.get\_errorMessage()**

---

Returns the error message of the latest error with the humidity sensor.

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the humidity sensor object

**humidity**→**get\_errorType()****YHumidity****humidity**→**errorType()****humidity.get\_errorType()**

Returns the numerical error code of the latest error with the humidity sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the humidity sensor object

**humidity→get\_friendlyName()**  
**humidity→friendlyName()**  
**humidity.get\_friendlyName()**

**YHumidity**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the humidity sensor using logical names (ex: `MyCustomName.relay1`)  
 On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**humidity→get\_functionDescriptor()**  
**humidity→functionDescriptor()**  
**humidity.get\_functionDescriptor()**

**YHumidity**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.



---

**humidity→get\_functionId()****YHumidity****humidity→functionId()humidity.get\_functionId()**

---

Returns the hardware identifier of the humidity sensor, without reference to the module.

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

For example `relay1`

**Returns :**

a string that identifies the humidity sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**humidity**→**get\_hardwareId()****YHumidity****humidity**→**hardwareId()****humidity.get\_hardwareId()**

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

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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

**Returns :**

a string that uniquely identifies the humidity sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**humidity→get\_highestValue()****YHumidity****humidity→highestValue()****humidity.get\_highestValue()**

Returns the maximal value observed for the humidity.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YHumidity <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the humidity

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**humidity→get\_logFrequency()****YHumidity****humidity→logFrequency()****humidity.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YHumidity <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

---

**humidity→get\_logicalName()****YHumidity****humidity→logicalName()humidity.get\_logicalName()**

---

Returns the logical name of the humidity sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YHumidity <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the humidity sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**humidity**→**get\_lowestValue()****YHumidity****humidity**→**lowestValue()****humidity.get\_lowestValue()**

Returns the minimal value observed for the humidity.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YHumidity <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the humidity

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**humidity→get\_module()****YHumidity****humidity→module()humidity.get\_module()**

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

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**humidity→get\_module\_async()****YHumidity****humidity→module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.



humidity→get\_recordedData()  
 humidity→recordedData()  
 humidity.get\_recordedData()

YHumidity

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YHumidity <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

#### Parameters :

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

#### Returns :

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**humidity→get\_reportFrequency()****YHumidity****humidity→reportFrequency()****humidity.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YHumidity <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**humidity→get\_resolution()****YHumidity****humidity→resolution()humidity.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YHumidity <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**humidity**→**get\_unit()****YHumidity****humidity**→**unit()****humidity.get\_unit()**

Returns the measuring unit for the humidity.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YHumidity <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the humidity

On failure, throws an exception or returns Y\_UNIT\_INVALID.

---

**humidity→get\_userdata()****YHumidity****humidity→userData()humidity.get\_userdata()**

---

Returns the value of the userData attribute, as previously stored using method set\_userdata.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**humidity→isOnline()humidity.isOnline()****YHumidity**

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

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

If there is a cached value for the humidity sensor 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 humidity sensor.

**Returns :**

`true` if the humidity sensor can be reached, and `false` otherwise

---

**humidity→isOnline\_async()****YHumidity**

---

Checks if the humidity sensor 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 humidity sensor 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.

**humidity**→**load()****humidity.load()****YHumidity**

Preloads the humidity sensor cache with a specified validity duration.

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

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 traffic 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.



## humidity→loadCalibrationPoints() humidity.loadCalibrationPoints()

YHumidity

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js      function loadCalibrationPoints( rawValues, refValues)
nodejs  function loadCalibrationPoints( rawValues, refValues)
php     function loadCalibrationPoints( &$rawValues, &$refValues)
cpp     int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m       -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )
cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)
cmd     YHumidity target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**humidity→load\_async()****YHumidity**

Preloads the humidity sensor 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 traffic 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.

**humidity→nextHumidity()humidity.nextHumidity()****YHumidity**

Continues the enumeration of humidity sensors started using `yFirstHumidity()`.

js	function <b>nextHumidity</b> ( )
nodejs	function <b>nextHumidity</b> ( )
php	function <b>nextHumidity</b> ( )
cpp	YHumidity * <b>nextHumidity</b> ( )
m	-(YHumidity*) <b>nextHumidity</b>
pas	function <b>nextHumidity</b> ( ): TYHumidity
vb	function <b>nextHumidity</b> ( ) As YHumidity
cs	YHumidity <b>nextHumidity</b> ( )
java	YHumidity <b>nextHumidity</b> ( )
py	def <b>nextHumidity</b> ( )

**Returns :**

a pointer to a `YHumidity` object, corresponding to a humidity sensor currently online, or a `null` pointer if there are no more humidity sensors to enumerate.

## humidity→registerTimedReportCallback() humidity.registerTimedReportCallback()

YHumidity

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YHumidityTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YHumidityTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYHumidityTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## humidity→registerValueCallback() humidity.registerValueCallback()

YHumidity

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YHumidityValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YHumidityValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYHumidityValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**humidity**→**set\_highestValue()****YHumidity****humidity**→**setHighestValue()****humidity.set\_highestValue()**

Changes the recorded maximal value observed for the humidity.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YHumidity <b>target</b> <b>set_highestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed for the humidity

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**humidity→set\_logFrequency()****YHumidity****humidity→setLogFrequency()****humidity.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YHumidity <b>target</b> <b>set_logFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**humidity→set\_logicalName()****YHumidity****humidity→setLogicalName()****humidity.set\_logicalName()**

Changes the logical name of the humidity sensor.

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

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 humidity sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.



humidity→set\_lowestValue()

YHumidity

humidity→setLowestValue()

humidity.set\_lowestValue()

Changes the recorded minimal value observed for the humidity.

js	function set_lowestValue( newval)
nodejs	function set_lowestValue( newval)
php	function set_lowestValue( \$newval)
cpp	int set_lowestValue( double newval)
m	-(int) setLowestValue : (double) newval
pas	function set_lowestValue( newval: double): integer
vb	function set_lowestValue( ByVal newval As Double) As Integer
cs	int set_lowestValue( double newval)
java	int set_lowestValue( double newval)
py	def set_lowestValue( newval)
cmd	YHumidity target set_lowestValue newval

#### Parameters :

**newval** a floating point number corresponding to the recorded minimal value observed for the humidity

#### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**humidity→set\_reportFrequency()****YHumidity****humidity→setReportFrequency()****humidity.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YHumidity <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**humidity→set\_resolution()****YHumidity****humidity→setResolution()humidity.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YHumidity <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**humidity→set\_userdata()****YHumidity****humidity→setUserData()humidity.set\_userdata()**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

## humidity→wait\_async()

## YHumidity

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.

## 3.21. Led function interface

Yoctopuce application programming interface allows you not only to drive the intensity of the led, but also to have it blink at various preset frequencies.

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

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

### Global functions

#### yFindLed(func)

Retrieves a led for a given identifier.

#### yFirstLed()

Starts the enumeration of leds currently accessible.

### YLed methods

#### led→describe()

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

#### led→get\_advertisedValue()

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

#### led→get\_blinking()

Returns the current led signaling mode.

#### led→get\_errorMessage()

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

#### led→get\_errorType()

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

#### led→get\_friendlyName()

Returns a global identifier of the led in the format MODULE\_NAME . FUNCTION\_NAME.

#### led→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### led→get\_functionId()

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

#### led→get\_hardwareId()

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

#### led→get\_logicalName()

Returns the logical name of the led.

#### led→get\_luminosity()

Returns the current led intensity (in per cent).

#### led→get\_module()

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

**led→get\_module\_async(callback, context)**

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

**led→get\_power()**

Returns the current led state.

**led→get\_userData()**

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

**led→isOnline()**

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

**led→isOnline\_async(callback, context)**

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

**led→load(msValidity)**

Preloads the led cache with a specified validity duration.

**led→load\_async(msValidity, callback, context)**

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

**led→nextLed()**

Continues the enumeration of leds started using `yFirstLed()`.

**led→registerValueCallback(callback)**

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

**led→set\_blinking(newval)**

Changes the current led signaling mode.

**led→set\_logicalName(newval)**

Changes the logical name of the led.

**led→set\_luminosity(newval)**

Changes the current led intensity (in per cent).

**led→set\_power(newval)**

Changes the state of the led.

**led→set\_userData(data)**

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

**led→wait\_async(callback, context)**

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

**YLed.FindLed()****YLed****yFindLed()****YLed.FindLed()**

Retrieves a led for a given identifier.

js	function <b>yFindLed</b> ( <b>func</b> )
nodejs	function <b>FindLed</b> ( <b>func</b> )
php	function <b>yFindLed</b> ( <b>\$func</b> )
cpp	YLed* <b>yFindLed</b> ( const string& <b>func</b> )
m	YLed* <b>yFindLed</b> ( NSString* <b>func</b> )
pas	function <b>yFindLed</b> ( <b>func</b> : string): TYLed
vb	function <b>yFindLed</b> ( ByVal <b>func</b> As String) As YLed
cs	YLed <b>FindLed</b> ( string <b>func</b> )
java	YLed <b>FindLed</b> ( String <b>func</b> )
py	def <b>FindLed</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the led is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YLed.isOnline()` to test if the led is indeed online at a given time. In case of ambiguity when looking for a led 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 led

**Returns :**

a YLed object allowing you to drive the led.



## YLed.FirstLed() yFirstLed()YLed.FirstLed()

YLed

Starts the enumeration of leds currently accessible.

js	function <b>yFirstLed</b> ( )
nodejs	function <b>FirstLed</b> ( )
php	function <b>yFirstLed</b> ( )
cpp	YLed* <b>yFirstLed</b> ( )
m	YLed* <b>yFirstLed</b> ( )
pas	function <b>yFirstLed</b> ( ): TYLed
vb	function <b>yFirstLed</b> ( ) As YLed
cs	YLed <b>FirstLed</b> ( )
java	YLed <b>FirstLed</b> ( )
py	def <b>FirstLed</b> ( )

Use the method `YLed.nextLed( )` to iterate on next leds.

### Returns :

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

**led→describe()****led.describe()****YLed**

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

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

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(BadCustomName.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 led (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**led**→**get\_advertisedValue()****YLed****led**→**advertisedValue()****led.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the led (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**led**→**get\_blinking()****YLed****led**→**blinking()****led.get\_blinking()**

Returns the current led signaling mode.

js	function <b>get_blinking</b> ( )
nodejs	function <b>get_blinking</b> ( )
php	function <b>get_blinking</b> ( )
cpp	Y_BLINKING_enum <b>get_blinking</b> ( )
m	-(Y_BLINKING_enum) blinking
pas	function <b>get_blinking</b> ( ): Integer
vb	function <b>get_blinking</b> ( ) As Integer
cs	int <b>get_blinking</b> ( )
java	int <b>get_blinking</b> ( )
py	def <b>get_blinking</b> ( )
cmd	YLed <b>target get_blinking</b>

**Returns :**

a value among Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL and Y\_BLINKING\_PANIC corresponding to the current led signaling mode

On failure, throws an exception or returns Y\_BLINKING\_INVALID.

**led**→**get\_errorMessage()****YLed****led**→**errorMessage()****led.get\_errorMessage()**

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

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

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

**Returns :**

a string corresponding to the latest error message that occurred while using the led object

**led**→**get\_errorType()****YLed****led**→**errorType()****led.get\_errorType()**

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

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the led object

**led**→**get\_friendlyName()****YLed****led**→**friendlyName()****led.get\_friendlyName()**

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

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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

**Returns :**

a string that uniquely identifies the led using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**led**→**get\_functionDescriptor()****YLed****led**→**functionDescriptor()****led.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

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

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.



**led**→**get\_functionId()****YLed****led**→**functionId()****led.get\_functionId()**

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

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

For example `relay1`

**Returns :**

a string that identifies the led (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**led**→**get\_hardwareId()****YLed****led**→**hardwareId()****led.get\_hardwareId()**

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

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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

**Returns :**

a string that uniquely identifies the led (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**led**→**get\_logicalName()****YLed****led**→**logicalName()****led.get\_logicalName()**

Returns the logical name of the led.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YLed <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the led. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**led**→**get\_luminosity()****YLed****led**→**luminosity()****led.get\_luminosity()**

Returns the current led intensity (in per cent).

js	function <b>get_luminosity</b> ( )
nodejs	function <b>get_luminosity</b> ( )
php	function <b>get_luminosity</b> ( )
cpp	int <b>get_luminosity</b> ( )
m	-(int) luminosity
pas	function <b>get_luminosity</b> ( ): LongInt
vb	function <b>get_luminosity</b> ( ) As Integer
cs	int <b>get_luminosity</b> ( )
java	int <b>get_luminosity</b> ( )
py	def <b>get_luminosity</b> ( )
cmd	YLed <b>target</b> <b>get_luminosity</b>

**Returns :**

an integer corresponding to the current led intensity (in per cent)

On failure, throws an exception or returns Y\_LUMINOSITY\_INVALID.

**led**→**get\_module()****YLed****led**→**module()****led.get\_module()**

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

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

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

**led**→**get\_module\_async()****YLed****led**→**module\_async()**

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

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**led**→**get\_power()****YLed****led**→**power()****led.get\_power()**

Returns the current led state.

js	function <b>get_power</b> ( )
nodejs	function <b>get_power</b> ( )
php	function <b>get_power</b> ( )
cpp	Y_POWER_enum <b>get_power</b> ( )
m	-(Y_POWER_enum) power
pas	function <b>get_power</b> ( ): Integer
vb	function <b>get_power</b> ( ) As Integer
cs	int <b>get_power</b> ( )
java	int <b>get_power</b> ( )
py	def <b>get_power</b> ( )
cmd	YLed <b>target</b> <b>get_power</b>

**Returns :**

either Y\_POWER\_OFF or Y\_POWER\_ON, according to the current led state

On failure, throws an exception or returns Y\_POWER\_INVALID.

**led**→**get\_userData()****YLed****led**→**userData()****led.userData()**

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

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

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.



**led→isOnline()****led.isOnline()****YLed**

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

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

If there is a cached value for the led 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 led.

**Returns :**

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

**led→isOnline\_async()****YLed**

Checks if the led 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 led 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.

**led→load()led.load()****YLed**

Preloads the led cache with a specified validity duration.

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

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 traffic 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.

**led→load\_async()****YLed**

Preloads the led 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 traffic 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.

**led→nextLed()****led.nextLed()****YLed**

Continues the enumeration of leds started using `yFirstLed()`.

js	function <b>nextLed</b> ( )
nodejs	function <b>nextLed</b> ( )
php	function <b>nextLed</b> ( )
cpp	YLed * <b>nextLed</b> ( )
m	-(YLed*) <b>nextLed</b>
pas	function <b>nextLed</b> ( ): TYLed
vb	function <b>nextLed</b> ( ) As YLed
cs	YLed <b>nextLed</b> ( )
java	YLed <b>nextLed</b> ( )
py	def <b>nextLed</b> ( )

**Returns :**

a pointer to a `YLed` object, corresponding to a led currently online, or a `null` pointer if there are no more leds to enumerate.

## led→registerValueCallback() led.registerValueCallback()

YLed

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

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YLedValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YLedValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYLedValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**led→set\_blinking()****YLed****led→setBlinking()led.set\_blinking()**

Changes the current led signaling mode.

js	function <b>set_blinking</b> ( <b>newval</b> )
nodejs	function <b>set_blinking</b> ( <b>newval</b> )
php	function <b>set_blinking</b> ( <b>\$newval</b> )
cpp	int <b>set_blinking</b> ( Y_BLINKING_enum <b>newval</b> )
m	-(int) setBlinking : (Y_BLINKING_enum) <b>newval</b>
pas	function <b>set_blinking</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_blinking</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_blinking</b> ( int <b>newval</b> )
java	int <b>set_blinking</b> ( int <b>newval</b> )
py	def <b>set_blinking</b> ( <b>newval</b> )
cmd	YLed <b>target set_blinking newval</b>

**Parameters :**

**newval** a value among Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL and Y\_BLINKING\_PANIC corresponding to the current led signaling mode

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**led**→**set\_logicalName()****YLed****led**→**setLogicalName()****led.set\_logicalName()**

Changes the logical name of the led.

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

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 led.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.



**led**→**set\_luminosity()****YLed****led**→**setLuminosity()****led.set\_luminosity()**

Changes the current led intensity (in per cent).

js	function <b>set_luminosity</b> ( <b>newval</b> )
nodejs	function <b>set_luminosity</b> ( <b>newval</b> )
php	function <b>set_luminosity</b> ( <b>\$newval</b> )
cpp	int <b>set_luminosity</b> ( int <b>newval</b> )
m	-(int) setLuminosity : (int) <b>newval</b>
pas	function <b>set_luminosity</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_luminosity</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_luminosity</b> ( int <b>newval</b> )
java	int <b>set_luminosity</b> ( int <b>newval</b> )
py	def <b>set_luminosity</b> ( <b>newval</b> )
cmd	YLed <b>target set_luminosity newval</b>

**Parameters :**

**newval** an integer corresponding to the current led intensity (in per cent)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**led**→**set\_power()****YLed****led**→**setPower()****led.set\_power()**

Changes the state of the led.

js	function <b>set_power</b> ( <b>newval</b> )
nodejs	function <b>set_power</b> ( <b>newval</b> )
php	function <b>set_power</b> ( <b>\$newval</b> )
cpp	int <b>set_power</b> ( Y_POWER_enum <b>newval</b> )
m	-(int) <b>setPower</b> : (Y_POWER_enum) <b>newval</b>
pas	function <b>set_power</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_power</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_power</b> ( int <b>newval</b> )
java	int <b>set_power</b> ( int <b>newval</b> )
py	def <b>set_power</b> ( <b>newval</b> )
cmd	YLed <b>target set_power newval</b>

**Parameters :**

**newval** either Y\_POWER\_OFF or Y\_POWER\_ON, according to the state of the led

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**led→set\_userdata()****YLed****led→setUserData()led.set\_userdata()**

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

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**led**→**wait\_async()****YLed**

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.

## 3.22. LightSensor function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

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

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

### Global functions

#### yFindLightSensor(func)

Retrieves a light sensor for a given identifier.

#### yFirstLightSensor()

Starts the enumeration of light sensors currently accessible.

### YLightSensor methods

#### lightsensor→calibrate(calibratedVal)

Changes the sensor-specific calibration parameter so that the current value matches a desired target (linear scaling).

#### lightsensor→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### lightsensor→describe()

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

#### lightsensor→get\_advertisedValue()

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

#### lightsensor→get\_currentRawValue()

Returns the unrounded and uncalibrated raw value returned by the sensor.

#### lightsensor→get\_currentValue()

Returns the current measure for the ambient light.

#### lightsensor→get\_errorMessage()

Returns the error message of the latest error with the light sensor.

#### lightsensor→get\_errorType()

Returns the numerical error code of the latest error with the light sensor.

#### lightsensor→get\_friendlyName()

Returns a global identifier of the light sensor in the format MODULE\_NAME . FUNCTION\_NAME.

#### lightsensor→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### lightsensor→get\_functionId()

Returns the hardware identifier of the light sensor, without reference to the module.

**lightsensor**→**get\_hardwareId()**

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

**lightsensor**→**get\_highestValue()**

Returns the maximal value observed for the ambient light.

**lightsensor**→**get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**lightsensor**→**get\_logicalName()**

Returns the logical name of the light sensor.

**lightsensor**→**get\_lowestValue()**

Returns the minimal value observed for the ambient light.

**lightsensor**→**get\_module()**

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

**lightsensor**→**get\_module\_async(callback, context)**

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

**lightsensor**→**get\_recordedData(startTime, endTime)**

Retrieves a `DataSet` object holding historical data for this sensor, for a specified time interval.

**lightsensor**→**get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**lightsensor**→**get\_resolution()**

Returns the resolution of the measured values.

**lightsensor**→**get\_unit()**

Returns the measuring unit for the ambient light.

**lightsensor**→**get\_userData()**

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

**lightsensor**→**isOnline()**

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

**lightsensor**→**isOnline\_async(callback, context)**

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

**lightsensor**→**load(msValidity)**

Preloads the light sensor cache with a specified validity duration.

**lightsensor**→**loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

**lightsensor**→**load\_async(msValidity, callback, context)**

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

**lightsensor**→**nextLightSensor()**

Continues the enumeration of light sensors started using `yFirstLightSensor()`.

**lightsensor**→**registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**lightsensor**→**registerValueCallback(callback)**

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

**lightsensor**→**set\_highestValue(newval)**

Changes the recorded maximal value observed for the ambient light.

**lightsensor**→**set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**lightsensor**→**set\_logicalName**(newval)

Changes the logical name of the light sensor.

**lightsensor**→**set\_lowestValue**(newval)

Changes the recorded minimal value observed for the ambient light.

**lightsensor**→**set\_reportFrequency**(newval)

Changes the timed value notification frequency for this function.

**lightsensor**→**set\_resolution**(newval)

Changes the resolution of the measured physical values.

**lightsensor**→**set\_userData**(data)

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

**lightsensor**→**wait\_async**(callback, context)

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

**YLightSensor.FindLightSensor()****YLightSensor****yFindLightSensor()YLightSensor.FindLightSensor()**

Retrieves a light sensor for a given identifier.

js	function <b>yFindLightSensor</b> ( <b>func</b> )
nodejs	function <b>FindLightSensor</b> ( <b>func</b> )
php	function <b>yFindLightSensor</b> ( <b>\$func</b> )
cpp	YLightSensor* <b>yFindLightSensor</b> ( const string& <b>func</b> )
m	YLightSensor* <b>yFindLightSensor</b> ( NSString* <b>func</b> )
pas	function <b>yFindLightSensor</b> ( <b>func</b> : string): TYLightSensor
vb	function <b>yFindLightSensor</b> ( ByVal <b>func</b> As String) As YLightSensor
cs	YLightSensor <b>FindLightSensor</b> ( string <b>func</b> )
java	YLightSensor <b>FindLightSensor</b> ( String <b>func</b> )
py	def <b>FindLightSensor</b> ( <b>func</b> )

The identifier can be specified using several formats:

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

This function does not require that the light sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YLightSensor.isOnline()` to test if the light sensor is indeed online at a given time. In case of ambiguity when looking for a light sensor 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 light sensor

**Returns :**

a `YLightSensor` object allowing you to drive the light sensor.



## YLightSensor.FirstLightSensor() yFirstLightSensor()YLightSensor.FirstLightSensor()

## YLightSensor

Starts the enumeration of light sensors currently accessible.

js	function <b>yFirstLightSensor</b> ( )
nodejs	function <b>FirstLightSensor</b> ( )
php	function <b>yFirstLightSensor</b> ( )
cpp	YLightSensor* <b>yFirstLightSensor</b> ( )
m	YLightSensor* <b>yFirstLightSensor</b> ( )
pas	function <b>yFirstLightSensor</b> ( ): TYLightSensor
vb	function <b>yFirstLightSensor</b> ( ) As YLightSensor
cs	YLightSensor <b>FirstLightSensor</b> ( )
java	YLightSensor <b>FirstLightSensor</b> ( )
py	def <b>FirstLightSensor</b> ( )

Use the method `YLightSensor.nextLightSensor( )` to iterate on next light sensors.

### Returns :

a pointer to a `YLightSensor` object, corresponding to the first light sensor currently online, or a `null` pointer if there are none.

**lightsensor→calibrate()**lightsensor.calibrate()**YLightSensor**

Changes the sensor-specific calibration parameter so that the current value matches a desired target (linear scaling).

js	function <b>calibrate</b> ( <b>calibratedVal</b> )
nodejs	function <b>calibrate</b> ( <b>calibratedVal</b> )
php	function <b>calibrate</b> ( <b>\$calibratedVal</b> )
cpp	int <b>calibrate</b> ( double <b>calibratedVal</b> )
m	-(int) <b>calibrate</b> : (double) <b>calibratedVal</b>
pas	function <b>calibrate</b> ( <b>calibratedVal</b> : double): integer
vb	function <b>calibrate</b> ( ByVal <b>calibratedVal</b> As Double) As Integer
cs	int <b>calibrate</b> ( double <b>calibratedVal</b> )
java	int <b>calibrate</b> ( double <b>calibratedVal</b> )
py	def <b>calibrate</b> ( <b>calibratedVal</b> )
cmd	YLightSensor <b>target</b> <b>calibrate</b> <b>calibratedVal</b>

**Parameters :**

**calibratedVal** the desired target value.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

## lightsensor→calibrateFromPoints() lightsensor.calibrateFromPoints()

## YLightSensor

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

js	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
nodejs	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
php	function <b>calibrateFromPoints</b> ( <b>\$rawValues</b> , <b>\$refValues</b> )
cpp	int <b>calibrateFromPoints</b> ( vector<double> <b>rawValues</b> , vector<double> <b>refValues</b> )
m	-(int) <b>calibrateFromPoints</b> : (NSMutableArray*) <b>rawValues</b> : (NSMutableArray*) <b>refValues</b>
pas	function <b>calibrateFromPoints</b> ( <b>rawValues</b> : TDoubleArray, <b>refValues</b> : TDoubleArray): LongInt
vb	procedure <b>calibrateFromPoints</b> ( )
cs	int <b>calibrateFromPoints</b> ( List<double> <b>rawValues</b> , List<double> <b>refValues</b> )
java	int <b>calibrateFromPoints</b> ( ArrayList<Double> <b>rawValues</b> , ArrayList<Double> <b>refValues</b> )
py	def <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
cmd	YLightSensor <b>target</b> <b>calibrateFromPoints</b> <b>rawValues</b> <b>refValues</b>

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

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

**lightsensor→describe()lightsensor.describe()****YLightSensor**

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

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

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(BadCustomName.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 light sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**lightsensor→get\_advertisedValue()****YLightSensor****lightsensor→advertisedValue()****lightsensor.get\_advertisedValue()**

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

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

**Returns :**

a string corresponding to the current value of the light sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**lightsensor→get\_currentRawValue()****YLightSensor****lightsensor→currentRawValue()****lightsensor.get\_currentRawValue()**

Returns the unrounded and uncalibrated raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YLightSensor <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the unrounded and uncalibrated raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**lightsensor→get\_currentValue()****YLightSensor****lightsensor→currentValue()****lightsensor.get\_currentValue()**

Returns the current measure for the ambient light.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YLightSensor <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the ambient light

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**lightsensor→get\_errorMessage()****YLightSensor****lightsensor→errorMessage()****lightsensor.get\_errorMessage()**

Returns the error message of the latest error with the light sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the light sensor object



**lightsensor**→**get\_errorType()****YLightSensor****lightsensor**→**errorType()****lightsensor.get\_errorType()**

Returns the numerical error code of the latest error with the light sensor.

<code>js</code>	<code>function get_errorType( )</code>
<code>nodejs</code>	<code>function get_errorType( )</code>
<code>php</code>	<code>function get_errorType( )</code>
<code>cpp</code>	<code>YRETCODE get_errorType( )</code>
<code>pas</code>	<code>function get_errorType( ): YRETCODE</code>
<code>vb</code>	<code>function get_errorType( ) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE get_errorType( )</code>
<code>java</code>	<code>int get_errorType( )</code>
<code>py</code>	<code>def get_errorType( )</code>

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 occurred while using the light sensor object

**lightsensor→get\_friendlyName()****YLightSensor****lightsensor→friendlyName()****lightsensor.get\_friendlyName()**

Returns a global identifier of the light sensor in the format `MODULE_NAME . FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the light sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the light sensor (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the light sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**lightsensor→get\_functionDescriptor()**  
**lightsensor→functionDescriptor()**  
**lightsensor.get\_functionDescriptor()**

**YLightSensor**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**lightsensor→get\_functionId()****YLightSensor****lightsensor→functionId()lightsensor.get\_functionId()**

Returns the hardware identifier of the light sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the light sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**lightsensor→get\_hardwareId()****YLightSensor****lightsensor→hardwareId()****lightsensor.get\_hardwareId()**

Returns the unique hardware identifier of the light sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the light sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the light sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**lightsensor→get\_highestValue()****YLightSensor****lightsensor→highestValue()****lightsensor.get\_highestValue()**

Returns the maximal value observed for the ambient light.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YLightSensor <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the ambient light

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**lightsensor→get\_logFrequency()****YLightSensor****lightsensor→logFrequency()****lightsensor.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YLightSensor <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**lightsensor→get\_logicalName()****YLightSensor****lightsensor→logicalName()****lightsensor.get\_logicalName()**

Returns the logical name of the light sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YLightSensor <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the light sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**lightsensor→get\_lowestValue()****YLightSensor****lightsensor→lowestValue()****lightsensor.get\_lowestValue()**

Returns the minimal value observed for the ambient light.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YLightSensor <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the ambient light

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**lightsensor→get\_module()****YLightSensor****lightsensor→module()lightsensor.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**lightsensor→get\_module\_async()****YLightSensor****lightsensor→module\_async()**

---

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**lightsensor→get\_recordedData()****YLightSensor****lightsensor→recordedData()****lightsensor.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
c++	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YLightSensor <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**lightsensor→get\_reportFrequency()****YLightSensor****lightsensor→reportFrequency()****lightsensor.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YLightSensor <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**lightsensor→get\_resolution()****YLightSensor****lightsensor→resolution()lightsensor.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YLightSensor <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**lightsensor**→**get\_unit()****YLightSensor****lightsensor**→**unit()****lightsensor.get\_unit()**

Returns the measuring unit for the ambient light.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YLightSensor <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the ambient light

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**lightsensor→get\_userData()****YLightSensor****lightsensor→userData()lightsensor.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.



**lightsensor→isOnline()lightsensor.isOnline()****YLightSensor**

Checks if the light sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the light sensor 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 light sensor.

**Returns :**

`true` if the light sensor can be reached, and `false` otherwise

**lightsensor→isOnline\_async()****YLightSensor**

Checks if the light sensor 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 light sensor 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.

**lightsensor→load()**lightsensor.load()**YLightSensor**

Preloads the light sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

## lightsensor→loadCalibrationPoints() lightsensor.loadCalibrationPoints()

YLightSensor

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YLightSensor target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**lightsensor→load\_async()****YLightSensor**

Preloads the light sensor 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 traffic 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.

**lightsensor→nextLightSensor()**  
**lightsensor.nextLightSensor()****YLightSensor**

Continues the enumeration of light sensors started using `yFirstLightSensor()`.

js	function <b>nextLightSensor</b> ( )
nodejs	function <b>nextLightSensor</b> ( )
php	function <b>nextLightSensor</b> ( )
c++	YLightSensor * <b>nextLightSensor</b> ( )
m	-(YLightSensor*) <b>nextLightSensor</b>
pas	function <b>nextLightSensor</b> ( ): TYLightSensor
vb	function <b>nextLightSensor</b> ( ) As YLightSensor
cs	YLightSensor <b>nextLightSensor</b> ( )
java	YLightSensor <b>nextLightSensor</b> ( )
py	def <b>nextLightSensor</b> ( )

**Returns :**

a pointer to a `YLightSensor` object, corresponding to a light sensor currently online, or a `null` pointer if there are no more light sensors to enumerate.

## lightsensor→registerTimedReportCallback() lightsensor.registerTimedReportCallback()

## YLightSensor

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YLightSensorTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YLightSensorTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYLightSensorTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## lightsensor→registerValueCallback()

### lightsensor.registerValueCallback()

YLightSensor

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YLightSensorValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YLightSensorValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYLightSensorValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.



**lightsensor→set\_highestValue()****YLightSensor****lightsensor→setHighestValue()****lightsensor.set\_highestValue()**

Changes the recorded maximal value observed for the ambient light.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YLightSensor <b>target</b> <b>set_highestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed for the ambient light

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**lightsensor→set\_logFrequency()****YLightSensor****lightsensor→setLogFrequency()****lightsensor.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YLightSensor <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**lightsensor→set\_logicalName()****YLightSensor****lightsensor→setLogicalName()****lightsensor.set\_logicalName()**

Changes the logical name of the light sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YLightSensor <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 light sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**lightsensor→set\_lowestValue()****YLightSensor****lightsensor→setLowestValue()****lightsensor.set\_lowestValue()**

Changes the recorded minimal value observed for the ambient light.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YLightSensor <b>target</b> <b>set_lowestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed for the ambient light

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**lightsensor→set\_reportFrequency()**  
**lightsensor→setReportFrequency()**  
**lightsensor.set\_reportFrequency()**

**YLightSensor**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YLightSensor <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**lightsensor→set\_resolution()****YLightSensor****lightsensor→setResolution()****lightsensor.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YLightSensor <b>target</b> <b>set_resolution</b> <b>newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**lightsensor→set\_userdata()****YLightSensor****lightsensor→setUserData()****lightsensor.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**lightsensor→wait\_async()****YLightSensor**

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.



## 3.23. Magnetometer function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<code>&lt;script type='text/javascript' src='yocto_magnetometer.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib'); var YMagnetometer = yoctolib.YMagnetometer;</code>
php	<code>require_once('yocto_magnetometer.php');</code>
c++	<code>#include "yocto_magnetometer.h"</code>
m	<code>#import "yocto_magnetometer.h"</code>
pas	<code>uses yocto_magnetometer;</code>
vb	<code>yocto_magnetometer.vb</code>
cs	<code>yocto_magnetometer.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YMagnetometer;</code>
py	<code>from yocto_magnetometer import *</code>

### Global functions

#### **yFindMagnetometer(func)**

Retrieves a magnetometer for a given identifier.

#### **yFirstMagnetometer()**

Starts the enumeration of magnetometers currently accessible.

### YMagnetometer methods

#### **magnetometer→calibrateFromPoints(rawValues, refValues)**

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### **magnetometer→describe()**

Returns a short text that describes unambiguously the instance of the magnetometer in the form `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### **magnetometer→get\_advertisedValue()**

Returns the current value of the magnetometer (no more than 6 characters).

#### **magnetometer→get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### **magnetometer→get\_currentValue()**

Returns the current value of the magnetic field.

#### **magnetometer→get\_errorMessage()**

Returns the error message of the latest error with the magnetometer.

#### **magnetometer→get\_errorType()**

Returns the numerical error code of the latest error with the magnetometer.

#### **magnetometer→get\_friendlyName()**

Returns a global identifier of the magnetometer in the format `MODULE_NAME . FUNCTION_NAME`.

#### **magnetometer→get\_functionDescriptor()**

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

#### **magnetometer→get\_functionId()**

Returns the hardware identifier of the magnetometer, without reference to the module.

#### **magnetometer→get\_hardwareId()**

Returns the unique hardware identifier of the magnetometer in the form `SERIAL . FUNCTIONID`.

**magnetometer→get\_highestValue()**

Returns the maximal value observed for the magnetic field since the device was started.

**magnetometer→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**magnetometer→get\_logicalName()**

Returns the logical name of the magnetometer.

**magnetometer→get\_lowestValue()**

Returns the minimal value observed for the magnetic field since the device was started.

**magnetometer→get\_module()**

Gets the YModule object for the device on which the function is located.

**magnetometer→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**magnetometer→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**magnetometer→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**magnetometer→get\_resolution()**

Returns the resolution of the measured values.

**magnetometer→get\_unit()**

Returns the measuring unit for the magnetic field.

**magnetometer→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**magnetometer→get\_xValue()**

Returns the X component of the magnetic field, as a floating point number.

**magnetometer→get\_yValue()**

Returns the Y component of the magnetic field, as a floating point number.

**magnetometer→get\_zValue()**

Returns the Z component of the magnetic field, as a floating point number.

**magnetometer→isOnline()**

Checks if the magnetometer is currently reachable, without raising any error.

**magnetometer→isOnline\_async(callback, context)**

Checks if the magnetometer is currently reachable, without raising any error (asynchronous version).

**magnetometer→load(msValidity)**

Preloads the magnetometer cache with a specified validity duration.

**magnetometer→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**magnetometer→load\_async(msValidity, callback, context)**

Preloads the magnetometer cache with a specified validity duration (asynchronous version).

**magnetometer→nextMagnetometer()**

Continues the enumeration of magnetometers started using yFirstMagnetometer().

**magnetometer→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**magnetometer→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**magnetometer→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**magnetometer→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**magnetometer→set\_logicalName(newval)**

Changes the logical name of the magnetometer.

**magnetometer→set\_lowestValue(newval)**

Changes the recorded minimal value observed.

**magnetometer→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**magnetometer→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**magnetometer→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**magnetometer→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YMagnetometer.FindMagnetometer() yFindMagnetometer() YMagnetometer.FindMagnetometer()

YMagnetometer

Retrieves a magnetometer for a given identifier.

js	function <b>yFindMagnetometer</b> ( <b>func</b> )
nodejs	function <b>FindMagnetometer</b> ( <b>func</b> )
php	function <b>yFindMagnetometer</b> ( <b>\$func</b> )
c++	YMagnetometer* <b>yFindMagnetometer</b> ( const string& <b>func</b> )
m	YMagnetometer* <b>yFindMagnetometer</b> ( NSString* <b>func</b> )
pas	function <b>yFindMagnetometer</b> ( <b>func</b> : string): TYMagnetometer
vb	function <b>yFindMagnetometer</b> ( ByVal <b>func</b> As String) As YMagnetometer
cs	YMagnetometer <b>FindMagnetometer</b> ( string <b>func</b> )
java	YMagnetometer <b>FindMagnetometer</b> ( String <b>func</b> )
py	def <b>FindMagnetometer</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the magnetometer is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YMagnetometer.isOnline()` to test if the magnetometer is indeed online at a given time. In case of ambiguity when looking for a magnetometer 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 magnetometer

### Returns :

a `YMagnetometer` object allowing you to drive the magnetometer.

## YMagnetometer.FirstMagnetometer() yFirstMagnetometer() YMagnetometer.FirstMagnetometer()

## YMagnetometer

Starts the enumeration of magnetometers currently accessible.

js	function <b>yFirstMagnetometer</b> ( )
nodejs	function <b>FirstMagnetometer</b> ( )
php	function <b>yFirstMagnetometer</b> ( )
cpp	YMagnetometer* <b>yFirstMagnetometer</b> ( )
m	YMagnetometer* <b>yFirstMagnetometer</b> ( )
pas	function <b>yFirstMagnetometer</b> ( ): TYMagnetometer
vb	function <b>yFirstMagnetometer</b> ( ) As YMagnetometer
cs	YMagnetometer <b>FirstMagnetometer</b> ( )
java	YMagnetometer <b>FirstMagnetometer</b> ( )
py	def <b>FirstMagnetometer</b> ( )

Use the method `YMagnetometer.nextMagnetometer( )` to iterate on next magnetometers.

### Returns :

a pointer to a `YMagnetometer` object, corresponding to the first magnetometer currently online, or a `null` pointer if there are none.

## magnetometer→calibrateFromPoints()

### magnetometer.calibrateFromPoints()

YMagnetometer

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YMagnetometer target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

#### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**magnetometer→describe()****magnetometer.describe()****YMagnetometer**

Returns a short text that describes unambiguously the instance of the magnetometer in the form `TYPE (NAME) = SERIAL . FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

More precisely, `TYPE` is the type of the function, `NAME` is 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(BadCustomName.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 magnetometer (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**magnetometer**→**get\_advertisedValue()****YMagnetometer****magnetometer**→**advertisedValue()****magnetometer.get\_advertisedValue()**

Returns the current value of the magnetometer (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the magnetometer (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.



**magnetometer→get\_currentRawValue()****YMagnetometer****magnetometer→currentRawValue()****magnetometer.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**magnetometer**→**get\_currentValue()****YMagnetometer****magnetometer**→**currentValue()****magnetometer.get\_currentValue()**

Returns the current value of the magnetic field.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the magnetic field

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**magnetometer→get\_errorMessage()****YMagnetometer****magnetometer→errorMessage()****magnetometer.get\_errorMessage()**

Returns the error message of the latest error with the magnetometer.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the magnetometer object

**magnetometer→get\_errorType()****YMagnetometer****magnetometer→errorType()****magnetometer.get\_errorType()**

Returns the numerical error code of the latest error with the magnetometer.

<code>js</code>	<code>function get_errorType( )</code>
<code>nodejs</code>	<code>function get_errorType( )</code>
<code>php</code>	<code>function get_errorType( )</code>
<code>cpp</code>	<code>YRETCODE get_errorType( )</code>
<code>pas</code>	<code>function get_errorType( ): YRETCODE</code>
<code>vb</code>	<code>function get_errorType( ) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE get_errorType( )</code>
<code>java</code>	<code>int get_errorType( )</code>
<code>py</code>	<code>def get_errorType( )</code>

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 occurred while using the magnetometer object

**magnetometer→get\_friendlyName()****YMagnetometer****magnetometer→friendlyName()****magnetometer.get\_friendlyName()**

Returns a global identifier of the magnetometer in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the magnetometer if they are defined, otherwise the serial number of the module and the hardware identifier of the magnetometer (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the magnetometer using logical names (ex: `MyCustomName.relay1`)

On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**magnetometer→get\_functionDescriptor()****YMagnetometer****magnetometer→functionDescriptor()****magnetometer.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**magnetometer→get\_functionId()****YMagnetometer****magnetometer→functionId()****magnetometer.get\_functionId()**

Returns the hardware identifier of the magnetometer, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the magnetometer (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**magnetometer→get\_hardwareId()**

**YMagnetometer**

**magnetometer→hardwareId()**

**magnetometer.get\_hardwareId()**

Returns the unique hardware identifier of the magnetometer in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the magnetometer. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the magnetometer (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.



**magnetometer→get\_highestValue()****YMagnetometer****magnetometer→highestValue()****magnetometer.get\_highestValue()**

Returns the maximal value observed for the magnetic field since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the magnetic field since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**magnetometer→get\_logFrequency()****YMagnetometer****magnetometer→logFrequency()****magnetometer.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**magnetometer→get\_logicalName()****YMagnetometer****magnetometer→logicalName()****magnetometer.get\_logicalName()**

Returns the logical name of the magnetometer.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the magnetometer. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**magnetometer**→**get\_lowestValue()****YMagnetometer****magnetometer**→**lowestValue()****magnetometer.get\_lowestValue()**

Returns the minimal value observed for the magnetic field since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the magnetic field since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**magnetometer→get\_module()****YMagnetometer****magnetometer→module()****magnetometer.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**magnetometer→get\_module\_async()**  
**magnetometer→module\_async()****YMagnetometer**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**magnetometer→get\_recordedData()****YMagnetometer****magnetometer→recordedData()****magnetometer.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YMagnetometer <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**magnetometer→get\_reportFrequency()****YMagnetometer****magnetometer→reportFrequency()****magnetometer.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.



**magnetometer→get\_resolution()****YMagnetometer****magnetometer→resolution()****magnetometer.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**magnetometer**→**get\_unit()****YMagnetometer****magnetometer**→**unit()****magnetometer.get\_unit()**

Returns the measuring unit for the magnetic field.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YMagnetometer <b>target get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the magnetic field

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**magnetometer→get\_userData()**  
**magnetometer→userData()**  
**magnetometer.get\_userData()**

**YMagnetometer**

Returns the value of the userData attribute, as previously stored using method set\_userData.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**magnetometer**→**get\_xValue()****YMagnetometer****magnetometer**→**xValue()****magnetometer.get\_xValue()**

Returns the X component of the magnetic field, as a floating point number.

js	function <b>get_xValue</b> ( )
nodejs	function <b>get_xValue</b> ( )
php	function <b>get_xValue</b> ( )
cpp	double <b>get_xValue</b> ( )
m	-(double) xValue
pas	function <b>get_xValue</b> ( ): double
vb	function <b>get_xValue</b> ( ) As Double
cs	double <b>get_xValue</b> ( )
java	double <b>get_xValue</b> ( )
py	def <b>get_xValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_xValue</b>

**Returns :**

a floating point number corresponding to the X component of the magnetic field, as a floating point number

On failure, throws an exception or returns Y\_XVALUE\_INVALID.

**magnetometer→get\_yValue()****YMagnetometer****magnetometer→yValue()****magnetometer.get\_yValue()**

Returns the Y component of the magnetic field, as a floating point number.

js	function <b>get_yValue</b> ( )
nodejs	function <b>get_yValue</b> ( )
php	function <b>get_yValue</b> ( )
cpp	double <b>get_yValue</b> ( )
m	-(double) yValue
pas	function <b>get_yValue</b> ( ): double
vb	function <b>get_yValue</b> ( ) As Double
cs	double <b>get_yValue</b> ( )
java	double <b>get_yValue</b> ( )
py	def <b>get_yValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_yValue</b>

**Returns :**

a floating point number corresponding to the Y component of the magnetic field, as a floating point number

On failure, throws an exception or returns Y\_YVALUE\_INVALID.

**magnetometer**→**get\_zValue()****YMagnetometer****magnetometer**→**zValue()****magnetometer.get\_zValue()**

Returns the Z component of the magnetic field, as a floating point number.

js	function <b>get_zValue</b> ( )
nodejs	function <b>get_zValue</b> ( )
php	function <b>get_zValue</b> ( )
cpp	double <b>get_zValue</b> ( )
m	-(double) zValue
pas	function <b>get_zValue</b> ( ): double
vb	function <b>get_zValue</b> ( ) As Double
cs	double <b>get_zValue</b> ( )
java	double <b>get_zValue</b> ( )
py	def <b>get_zValue</b> ( )
cmd	YMagnetometer <b>target</b> <b>get_zValue</b>

**Returns :**

a floating point number corresponding to the Z component of the magnetic field, as a floating point number

On failure, throws an exception or returns Y\_ZVALUE\_INVALID.

**magnetometer**→**isOnline()****magnetometer.isOnline()****YMagnetometer**

Checks if the magnetometer is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the magnetometer 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 magnetometer.

**Returns :**

`true` if the magnetometer can be reached, and `false` otherwise

**magnetometer→isOnline\_async()****YMagnetometer**

Checks if the magnetometer 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 magnetometer 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.



**magnetometer→load()magnetometer.load()****YMagnetometer**

Preloads the magnetometer cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

**magnetometer→loadCalibrationPoints()****YMagnetometer****magnetometer.loadCalibrationPoints()**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YMagnetometer target loadCalibrationPoints rawValues refValues

```

**Parameters :**

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**magnetometer→load\_async()****YMagnetometer**

Preloads the magnetometer 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 traffic 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.

**magnetometer**→**nextMagnetometer()**  
**magnetometer.nextMagnetometer()**

**YMagnetometer**

Continues the enumeration of magnetometers started using `yFirstMagnetometer()`.

js	function <b>nextMagnetometer</b> ( )
nodejs	function <b>nextMagnetometer</b> ( )
php	function <b>nextMagnetometer</b> ( )
cpp	YMagnetometer * <b>nextMagnetometer</b> ( )
m	-(YMagnetometer*) <b>nextMagnetometer</b>
pas	function <b>nextMagnetometer</b> ( ): TYMagnetometer
vb	function <b>nextMagnetometer</b> ( ) As YMagnetometer
cs	YMagnetometer <b>nextMagnetometer</b> ( )
java	YMagnetometer <b>nextMagnetometer</b> ( )
py	def <b>nextMagnetometer</b> ( )

**Returns :**

a pointer to a `YMagnetometer` object, corresponding to a magnetometer currently online, or a `null` pointer if there are no more magnetometers to enumerate.

## magnetometer→registerTimedReportCallback() magnetometer.registerTimedReportCallback()

## YMagnetometer

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YMagnetometerTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YMagnetometerTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYMagnetometerTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

**magnetometer→registerValueCallback()****YMagnetometer****magnetometer.registerValueCallback()**

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YMagnetometerValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YMagnetometerValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYMagnetometerValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**magnetometer→set\_highestValue()**  
**magnetometer→setHighestValue()**  
**magnetometer.set\_highestValue()**

**YMagnetometer**

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YMagnetometer <b>target</b> <b>set_highestValue</b> <b>newval</b>

#### Parameters :

**newval** a floating point number corresponding to the recorded maximal value observed

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**magnetometer→set\_logFrequency()**  
**magnetometer→setLogFrequency()**  
**magnetometer.set\_logFrequency()**

**YMagnetometer**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YMagnetometer <b>target</b> <b>set_logFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the datalogger recording frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**magnetometer→set\_logicalName()****YMagnetometer****magnetometer→setLogicalName()****magnetometer.set\_logicalName()**

Changes the logical name of the magnetometer.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YMagnetometer <b>target set_logicalName newval</b>

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 magnetometer.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**magnetometer**→**set\_lowestValue()****YMagnetometer****magnetometer**→**setLowestValue()****magnetometer.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YMagnetometer <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**magnetometer→set\_reportFrequency()**  
**magnetometer→setReportFrequency()**  
**magnetometer.set\_reportFrequency()**

**YMagnetometer**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YMagnetometer <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**magnetometer→set\_resolution()****YMagnetometer****magnetometer→setResolution()****magnetometer.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YMagnetometer <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**magnetometer→set\_userdata()**  
**magnetometer→setUserData()**  
**magnetometer.set\_userdata()**

**YMagnetometer**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**magnetometer→wait\_async()****YMagnetometer**

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.

## 3.24. Measured value

YMeasure objects are used within the API to represent a value measured at a specified time. These objects are used in particular in conjunction with the YDataSet class.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_api.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;
php	require_once('yocto_api.php');
cpp	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
java	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *

### YMeasure methods

#### **measure→get\_averageValue()**

Returns the average value observed during the time interval covered by this measure.

#### **measure→get\_endTimeUTC()**

Returns the end time of the measure, relative to the Jan 1, 1970 UTC (Unix timestamp).

#### **measure→get\_maxValue()**

Returns the largest value observed during the time interval covered by this measure.

#### **measure→get\_minValue()**

Returns the smallest value observed during the time interval covered by this measure.

#### **measure→get\_startTimeUTC()**

Returns the start time of the measure, relative to the Jan 1, 1970 UTC (Unix timestamp).

**measure**→**get\_averageValue()****YMeasure****measure**→**averageValue()****measure.get\_averageValue()**

Returns the average value observed during the time interval covered by this measure.

js	function <b>get_averageValue</b> ( )
nodejs	function <b>get_averageValue</b> ( )
php	function <b>get_averageValue</b> ( )
cpp	double <b>get_averageValue</b> ( )
m	-(double) averageValue
pas	function <b>get_averageValue</b> ( ): double
vb	function <b>get_averageValue</b> ( ) As Double
cs	double <b>get_averageValue</b> ( )
java	double <b>get_averageValue</b> ( )
py	def <b>get_averageValue</b> ( )

**Returns :**

a floating-point number corresponding to the average value observed.



**measure**→**get\_endTimeUTC()****YMeasure****measure**→**endTimeUTC()****measure.get\_endTimeUTC()**

Returns the end time of the measure, relative to the Jan 1, 1970 UTC (Unix timestamp).

js	function <b>get_endTimeUTC</b> ( )
nodejs	function <b>get_endTimeUTC</b> ( )
php	function <b>get_endTimeUTC</b> ( )
cpp	double <b>get_endTimeUTC</b> ( )
m	-(double) endTimeUTC
pas	function <b>get_endTimeUTC</b> ( ): double
vb	function <b>get_endTimeUTC</b> ( ) As Double
cs	double <b>get_endTimeUTC</b> ( )
java	double <b>get_endTimeUTC</b> ( )
py	def <b>get_endTimeUTC</b> ( )

When the recording rate is higher then 1 sample per second, the timestamp may have a fractional part.

**Returns :**

an floating point number corresponding to the number of seconds between the Jan 1, 1970 UTC and the end of this measure.

**measure**→**get\_maxValue()****YMeasure****measure**→**maxValue()****measure.get\_maxValue()**

Returns the largest value observed during the time interval covered by this measure.

js	function <b>get_maxValue</b> ( )
nodejs	function <b>get_maxValue</b> ( )
php	function <b>get_maxValue</b> ( )
cpp	double <b>get_maxValue</b> ( )
m	-(double) <b>maxValue</b>
pas	function <b>get_maxValue</b> ( ): double
vb	function <b>get_maxValue</b> ( ) As Double
cs	double <b>get_maxValue</b> ( )
java	double <b>get_maxValue</b> ( )
py	def <b>get_maxValue</b> ( )

**Returns :**

a floating-point number corresponding to the largest value observed.

**measure**→**get\_minValue()****YMeasure****measure**→**minValue()****measure.get\_minValue()**

Returns the smallest value observed during the time interval covered by this measure.

js	function <b>get_minValue</b> ( )
nodejs	function <b>get_minValue</b> ( )
php	function <b>get_minValue</b> ( )
cpp	double <b>get_minValue</b> ( )
m	-(double) minValue
pas	function <b>get_minValue</b> ( ): double
vb	function <b>get_minValue</b> ( ) As Double
cs	double <b>get_minValue</b> ( )
java	double <b>get_minValue</b> ( )
py	def <b>get_minValue</b> ( )

**Returns :**

a floating-point number corresponding to the smallest value observed.

**measure**→**get\_startTimeUTC()****YMeasure****measure**→**startTimeUTC()****measure.get\_startTimeUTC()**

Returns the start time of the measure, relative to the Jan 1, 1970 UTC (Unix timestamp).

js	function <b>get_startTimeUTC</b> ( )
nodejs	function <b>get_startTimeUTC</b> ( )
php	function <b>get_startTimeUTC</b> ( )
cpp	double <b>get_startTimeUTC</b> ( )
m	-(double) startTimeUTC
pas	function <b>get_startTimeUTC</b> ( ): double
vb	function <b>get_startTimeUTC</b> ( ) As Double
cs	double <b>get_startTimeUTC</b> ( )
java	double <b>get_startTimeUTC</b> ( )
py	def <b>get_startTimeUTC</b> ( )

When the recording rate is higher than 1 sample per second, the timestamp may have a fractional part.

**Returns :**

an floating point number corresponding to the number of seconds between the Jan 1, 1970 UTC and the beginning of this measure.

## 3.25. Module control interface

This interface is identical for all Yoctopuce USB modules. It can be used to control the module global parameters, and to enumerate the functions provided by each module.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_api.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;
php	require_once('yocto_api.php');
c++	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
java	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *

### Global functions

#### yFindModule(func)

Allows you to find a module from its serial number or from its logical name.

#### yFirstModule()

Starts the enumeration of modules currently accessible.

### YModule methods

#### module→describe()

Returns a descriptive text that identifies the module.

#### module→download(pathname)

Downloads the specified built-in file and returns a binary buffer with its content.

#### module→functionCount()

Returns the number of functions (beside the "module" interface) available on the module.

#### module→functionId(functionIndex)

Retrieves the hardware identifier of the *n*th function on the module.

#### module→functionName(functionIndex)

Retrieves the logical name of the *n*th function on the module.

#### module→functionValue(functionIndex)

Retrieves the advertised value of the *n*th function on the module.

#### module→get\_beacon()

Returns the state of the localization beacon.

#### module→get\_errorMessage()

Returns the error message of the latest error with this module object.

#### module→get\_errorType()

Returns the numerical error code of the latest error with this module object.

#### module→get\_firmwareRelease()

Returns the version of the firmware embedded in the module.

#### module→get\_hardwareId()

Returns the unique hardware identifier of the module.

#### module→get\_icon2d()

### 3. Reference

	Returns the icon of the module.
<b>module</b> → <b>get_lastLogs()</b>	Returns a string with last logs of the module.
<b>module</b> → <b>get_logicalName()</b>	Returns the logical name of the module.
<b>module</b> → <b>get_luminosity()</b>	Returns the luminosity of the module informative leds (from 0 to 100).
<b>module</b> → <b>get_persistentSettings()</b>	Returns the current state of persistent module settings.
<b>module</b> → <b>get_productId()</b>	Returns the USB device identifier of the module.
<b>module</b> → <b>get_productName()</b>	Returns the commercial name of the module, as set by the factory.
<b>module</b> → <b>get_productRelease()</b>	Returns the hardware release version of the module.
<b>module</b> → <b>get_rebootCountdown()</b>	Returns the remaining number of seconds before the module restarts, or zero when no reboot has been scheduled.
<b>module</b> → <b>get_serialNumber()</b>	Returns the serial number of the module, as set by the factory.
<b>module</b> → <b>get_upTime()</b>	Returns the number of milliseconds spent since the module was powered on.
<b>module</b> → <b>get_usbBandwidth()</b>	Returns the number of USB interfaces used by the module.
<b>module</b> → <b>get_usbCurrent()</b>	Returns the current consumed by the module on the USB bus, in milli-amps.
<b>module</b> → <b>get_userData()</b>	Returns the value of the userData attribute, as previously stored using method <code>set_userData</code> .
<b>module</b> → <b>isOnline()</b>	Checks if the module is currently reachable, without raising any error.
<b>module</b> → <b>isOnline_async(callback, context)</b>	Checks if the module is currently reachable, without raising any error.
<b>module</b> → <b>load(msValidity)</b>	Preloads the module cache with a specified validity duration.
<b>module</b> → <b>load_async(msValidity, callback, context)</b>	Preloads the module cache with a specified validity duration (asynchronous version).
<b>module</b> → <b>nextModule()</b>	Continues the module enumeration started using <code>yFirstModule()</code> .
<b>module</b> → <b>reboot(secBeforeReboot)</b>	Schedules a simple module reboot after the given number of seconds.
<b>module</b> → <b>registerLogCallback(callback)</b>	todo
<b>module</b> → <b>revertFromFlash()</b>	Reloads the settings stored in the nonvolatile memory, as when the module is powered on.
<b>module</b> → <b>saveToFlash()</b>	Saves current settings in the nonvolatile memory of the module.

**module→set\_beacon(newval)**

Turns on or off the module localization beacon.

**module→set\_logicalName(newval)**

Changes the logical name of the module.

**module→set\_luminosity(newval)**

Changes the luminosity of the module informative leds.

**module→set\_usbBandwidth(newval)**

Changes the number of USB interfaces used by the module.

**module→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**module→triggerFirmwareUpdate(secBeforeReboot)**

Schedules a module reboot into special firmware update mode.

**module→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YModule.FindModule() yFindModule()YModule.FindModule()

YModule

Allows you to find a module from its serial number or from its logical name.

js	function <b>yFindModule</b> ( <b>func</b> )
nodejs	function <b>FindModule</b> ( <b>func</b> )
php	function <b>yFindModule</b> ( <b>\$func</b> )
cpp	YModule* <b>yFindModule</b> ( string <b>func</b> )
m	+(YModule*) <b>yFindModule</b> : (NSString*) <b>func</b>
pas	function <b>yFindModule</b> ( <b>func</b> : string): TYModule
vb	function <b>yFindModule</b> ( ByVal <b>func</b> As String) As YModule
cs	YModule <b>FindModule</b> ( string <b>func</b> )
java	YModule <b>FindModule</b> ( String <b>func</b> )
py	def <b>FindModule</b> ( <b>func</b> )

This function does not require that the module is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YModule.isOnline()` to test if the module is indeed online at a given time. In case of ambiguity when looking for a module 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 containing either the serial number or the logical name of the desired module

### Returns :

a `YModule` object allowing you to drive the module or get additional information on the module.



**YModule.FirstModule()****YModule****yFirstModule()YModule.FirstModule()**

Starts the enumeration of modules currently accessible.

js	function <b>yFirstModule</b> ( )
nodejs	function <b>FirstModule</b> ( )
php	function <b>yFirstModule</b> ( )
cpp	YModule* <b>yFirstModule</b> ( )
m	YModule* <b>yFirstModule</b> ( )
pas	function <b>yFirstModule</b> ( ): TYModule
vb	function <b>yFirstModule</b> ( ) As YModule
cs	YModule <b>FirstModule</b> ( )
java	YModule <b>FirstModule</b> ( )
py	def <b>FirstModule</b> ( )

Use the method `YModule.nextModule( )` to iterate on the next modules.

**Returns :**

a pointer to a `YModule` object, corresponding to the first module currently online, or a `null` pointer if there are none.

**module**→**describe()****module.describe()****YModule**

Returns a descriptive text that identifies the module.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

The text may include either the logical name or the serial number of the module.

**Returns :**

a string that describes the module

**module→download()module.download()****YModule**

Downloads the specified built-in file and returns a binary buffer with its content.

js	function <b>download</b> ( <b>pathname</b> )
nodejs	function <b>download</b> ( <b>pathname</b> )
php	function <b>download</b> ( <b>\$pathname</b> )
cpp	string <b>download</b> ( string <b>pathname</b> )
m	-(NSData*) <b>download</b> : (NSString*) <b>pathname</b>
pas	function <b>download</b> ( <b>pathname</b> : string): TByteArray
vb	function <b>download</b> ( ) As Byte
py	def <b>download</b> ( <b>pathname</b> )
cmd	YModule <b>target download</b> <b>pathname</b>

**Parameters :**

**pathname** name of the new file to load

**Returns :**

a binary buffer with the file content

On failure, throws an exception or returns an empty content.

module→functionCount()module.functionCount()

YModule

Returns the number of functions (beside the "module" interface) available on the module.

js	function functionCount( )
nodejs	function functionCount( )
php	function functionCount( )
cpp	int functionCount( )
m	-(int) functionCount
pas	function functionCount( ): integer
vb	function functionCount( ) As Integer
cs	int functionCount( )
py	def functionCount( )

**Returns :**  
the number of functions on the module

On failure, throws an exception or returns a negative error code.

**module**→**functionId()****module.functionId()****YModule**

Retrieves the hardware identifier of the *n*th function on the module.

js	function <b>functionId</b> ( <b>functionIndex</b> )
nodejs	function <b>functionId</b> ( <b>functionIndex</b> )
php	function <b>functionId</b> ( <b>\$functionIndex</b> )
cpp	string <b>functionId</b> ( int <b>functionIndex</b> )
m	-(NSString*) <b>functionId</b> : (int) <b>functionIndex</b>
pas	function <b>functionId</b> ( <b>functionIndex</b> : integer): string
vb	function <b>functionId</b> ( ByVal <b>functionIndex</b> As Integer) As String
cs	string <b>functionId</b> ( int <b>functionIndex</b> )
py	def <b>functionId</b> ( <b>functionIndex</b> )

**Parameters :**

**functionIndex** the index of the function for which the information is desired, starting at 0 for the first function.

**Returns :**

a string corresponding to the unambiguous hardware identifier of the requested module function

On failure, throws an exception or returns an empty string.

**module**→**functionName()****module.functionName()****YModule**

Retrieves the logical name of the  $n$ th function on the module.

js	function <b>functionName</b> ( <b>functionIndex</b> )
nodejs	function <b>functionName</b> ( <b>functionIndex</b> )
php	function <b>functionName</b> ( <b>\$functionIndex</b> )
cpp	string <b>functionName</b> ( int <b>functionIndex</b> )
m	-(NSString*) <b>functionName</b> : (int) <b>functionIndex</b>
pas	function <b>functionName</b> ( <b>functionIndex</b> : integer): string
vb	function <b>functionName</b> ( ByVal <b>functionIndex</b> As Integer) As String
cs	string <b>functionName</b> ( int <b>functionIndex</b> )
py	def <b>functionName</b> ( <b>functionIndex</b> )

**Parameters :**

**functionIndex** the index of the function for which the information is desired, starting at 0 for the first function.

**Returns :**

a string corresponding to the logical name of the requested module function

On failure, throws an exception or returns an empty string.

**module**→**functionValue()****module.functionValue()****YModule**

Retrieves the advertised value of the *n*th function on the module.

js	function <b>functionValue</b> ( <b>functionIndex</b> )
nodejs	function <b>functionValue</b> ( <b>functionIndex</b> )
php	function <b>functionValue</b> ( <b>\$functionIndex</b> )
cpp	string <b>functionValue</b> ( int <b>functionIndex</b> )
m	-(NSString*) <b>functionValue</b> : (int) <b>functionIndex</b>
pas	function <b>functionValue</b> ( <b>functionIndex</b> : integer): string
vb	function <b>functionValue</b> ( ByVal <b>functionIndex</b> As Integer) As String
cs	string <b>functionValue</b> ( int <b>functionIndex</b> )
py	def <b>functionValue</b> ( <b>functionIndex</b> )

**Parameters :**

**functionIndex** the index of the function for which the information is desired, starting at 0 for the first function.

**Returns :**

a short string (up to 6 characters) corresponding to the advertised value of the requested module function

On failure, throws an exception or returns an empty string.

**module**→**get\_beacon()****YModule****module**→**beacon()****module.get\_beacon()**

Returns the state of the localization beacon.

js	function <b>get_beacon</b> ( )
nodejs	function <b>get_beacon</b> ( )
php	function <b>get_beacon</b> ( )
cpp	Y_BEACON_enum <b>get_beacon</b> ( )
m	-(Y_BEACON_enum) beacon
pas	function <b>get_beacon</b> ( ): Integer
vb	function <b>get_beacon</b> ( ) As Integer
cs	int <b>get_beacon</b> ( )
java	int <b>get_beacon</b> ( )
py	def <b>get_beacon</b> ( )
cmd	YModule <b>target</b> <b>get_beacon</b>

**Returns :**

either Y\_BEACON\_OFF or Y\_BEACON\_ON, according to the state of the localization beacon

On failure, throws an exception or returns Y\_BEACON\_INVALID.



---

**module**→**get\_errorMessage()****YModule****module**→**errorMessage()****module.errorMessage()**

---

Returns the error message of the latest error with this module object.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using this module object

**module**→**get\_errorType()****YModule****module**→**errorType()****module.get\_errorType()**

Returns the numerical error code of the latest error with this module object.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using this module object

**module**→**get\_firmwareRelease()**

**YModule**

**module**→**firmwareRelease()**

**module.get\_firmwareRelease()**

Returns the version of the firmware embedded in the module.

js	function <b>get_firmwareRelease</b> ( )
nodejs	function <b>get_firmwareRelease</b> ( )
php	function <b>get_firmwareRelease</b> ( )
cpp	string <b>get_firmwareRelease</b> ( )
m	-(NSString*) firmwareRelease
pas	function <b>get_firmwareRelease</b> ( ): string
vb	function <b>get_firmwareRelease</b> ( ) As String
cs	string <b>get_firmwareRelease</b> ( )
java	String <b>get_firmwareRelease</b> ( )
py	def <b>get_firmwareRelease</b> ( )
cmd	YModule <b>target</b> <b>get_firmwareRelease</b>

#### Returns :

a string corresponding to the version of the firmware embedded in the module

On failure, throws an exception or returns Y\_FIRMWARERELEASE\_INVALID.

**module**→**get\_hardwareId()****YModule****module**→**hardwareId()****module.get\_hardwareId()**

Returns the unique hardware identifier of the module.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is made of the device serial number followed by string ".module".

**Returns :**

a string that uniquely identifies the module

---

**module**→**get\_icon2d()****YModule****module**→**icon2d()****module.get\_icon2d()**

---

Returns the icon of the module.

js	function <b>get_icon2d</b> ( )
nodejs	function <b>get_icon2d</b> ( )
php	function <b>get_icon2d</b> ( )
cpp	string <b>get_icon2d</b> ( )
m	-(NSData*) icon2d
pas	function <b>get_icon2d</b> ( ): TByteArray
vb	function <b>get_icon2d</b> ( ) As Byte
py	def <b>get_icon2d</b> ( )
cmd	YModule <b>target</b> <b>get_icon2d</b>

The icon is a PNG image and does not exceeds 1536 bytes.

**Returns :**

a binary buffer with module icon, in png format.

**module**→**get\_lastLogs()****YModule****module**→**lastLogs()****module.get\_lastLogs()**

Returns a string with last logs of the module.

js	function <b>get_lastLogs</b> ( )
nodejs	function <b>get_lastLogs</b> ( )
php	function <b>get_lastLogs</b> ( )
cpp	string <b>get_lastLogs</b> ( )
m	-(NSString*) lastLogs
pas	function <b>get_lastLogs</b> ( ): string
vb	function <b>get_lastLogs</b> ( ) As String
cs	string <b>get_lastLogs</b> ( )
java	String <b>get_lastLogs</b> ( )
py	def <b>get_lastLogs</b> ( )
cmd	YModule <b>target</b> <b>get_lastLogs</b>

This method return only logs that are still in the module.

**Returns :**

a string with last logs of the module.

**module**→**get\_logicalName()****YModule****module**→**logicalName()****module.get\_logicalName()**

Returns the logical name of the module.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YModule <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the module

On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**module**→**get\_luminosity()****YModule****module**→**luminosity()****module.get\_luminosity()**

Returns the luminosity of the module informative leds (from 0 to 100).

js	function <b>get_luminosity</b> ( )
nodejs	function <b>get_luminosity</b> ( )
php	function <b>get_luminosity</b> ( )
cpp	int <b>get_luminosity</b> ( )
m	-(int) luminosity
pas	function <b>get_luminosity</b> ( ): LongInt
vb	function <b>get_luminosity</b> ( ) As Integer
cs	int <b>get_luminosity</b> ( )
java	int <b>get_luminosity</b> ( )
py	def <b>get_luminosity</b> ( )
cmd	YModule <b>target</b> <b>get_luminosity</b>

**Returns :**

an integer corresponding to the luminosity of the module informative leds (from 0 to 100)

On failure, throws an exception or returns Y\_LUMINOSITY\_INVALID.



**module→get\_persistentSettings()****YModule****module→persistentSettings()****module.get\_persistentSettings()**

Returns the current state of persistent module settings.

js	function <b>get_persistentSettings</b> ( )
nodejs	function <b>get_persistentSettings</b> ( )
php	function <b>get_persistentSettings</b> ( )
cpp	Y_PERSISTENTSETTINGS_enum <b>get_persistentSettings</b> ( )
m	-(Y_PERSISTENTSETTINGS_enum) persistentSettings
pas	function <b>get_persistentSettings</b> ( ): Integer
vb	function <b>get_persistentSettings</b> ( ) As Integer
cs	int <b>get_persistentSettings</b> ( )
java	int <b>get_persistentSettings</b> ( )
py	def <b>get_persistentSettings</b> ( )
cmd	YModule <b>target</b> <b>get_persistentSettings</b>

**Returns :**

a value among Y\_PERSISTENTSETTINGS\_LOADED, Y\_PERSISTENTSETTINGS\_SAVED and Y\_PERSISTENTSETTINGS\_MODIFIED corresponding to the current state of persistent module settings

On failure, throws an exception or returns Y\_PERSISTENTSETTINGS\_INVALID.

**module**→**get\_productId()****YModule****module**→**productId()****module.get\_productId()**

Returns the USB device identifier of the module.

js	function <b>get_productId</b> ( )
nodejs	function <b>get_productId</b> ( )
php	function <b>get_productId</b> ( )
cpp	int <b>get_productId</b> ( )
m	-(int) productId
pas	function <b>get_productId</b> ( ): LongInt
vb	function <b>get_productId</b> ( ) As Integer
cs	int <b>get_productId</b> ( )
java	int <b>get_productId</b> ( )
py	def <b>get_productId</b> ( )
cmd	YModule <b>target</b> <b>get_productId</b>

**Returns :**

an integer corresponding to the USB device identifier of the module

On failure, throws an exception or returns Y\_PRODUCTID\_INVALID.

---

**module**→**get\_productName()**
**YModule****module**→**productName()****module.get\_productName()**


---

 Returns the commercial name of the module, as set by the factory.

js	function <b>get_productName</b> ( )
nodejs	function <b>get_productName</b> ( )
php	function <b>get_productName</b> ( )
cpp	string <b>get_productName</b> ( )
m	-(NSString*) productName
pas	function <b>get_productName</b> ( ): string
vb	function <b>get_productName</b> ( ) As String
cs	string <b>get_productName</b> ( )
java	String <b>get_productName</b> ( )
py	def <b>get_productName</b> ( )
cmd	YModule <b>target</b> <b>get_productName</b>

**Returns :**

a string corresponding to the commercial name of the module, as set by the factory

On failure, throws an exception or returns Y\_PRODUCTNAME\_INVALID.

**module**→**get\_productRelease()****YModule****module**→**productRelease()****module.get\_productRelease()**

Returns the hardware release version of the module.

js	function <b>get_productRelease</b> ( )
nodejs	function <b>get_productRelease</b> ( )
php	function <b>get_productRelease</b> ( )
cpp	int <b>get_productRelease</b> ( )
m	-(int) productRelease
pas	function <b>get_productRelease</b> ( ): LongInt
vb	function <b>get_productRelease</b> ( ) As Integer
cs	int <b>get_productRelease</b> ( )
java	int <b>get_productRelease</b> ( )
py	def <b>get_productRelease</b> ( )
cmd	YModule <b>target</b> <b>get_productRelease</b>

**Returns :**

an integer corresponding to the hardware release version of the module

On failure, throws an exception or returns Y\_PRODUCTRELEASE\_INVALID.

**module→get\_rebootCountdown()**

**YModule**

**module→rebootCountdown()**

**module.get\_rebootCountdown()**

Returns the remaining number of seconds before the module restarts, or zero when no reboot has been scheduled.

js	function <b>get_rebootCountdown</b> ( )
nodejs	function <b>get_rebootCountdown</b> ( )
php	function <b>get_rebootCountdown</b> ( )
cpp	int <b>get_rebootCountdown</b> ( )
m	-(int) rebootCountdown
pas	function <b>get_rebootCountdown</b> ( ): LongInt
vb	function <b>get_rebootCountdown</b> ( ) As Integer
cs	int <b>get_rebootCountdown</b> ( )
java	int <b>get_rebootCountdown</b> ( )
py	def <b>get_rebootCountdown</b> ( )
cmd	YModule <b>target</b> <b>get_rebootCountdown</b>

#### Returns :

an integer corresponding to the remaining number of seconds before the module restarts, or zero when no reboot has been scheduled

On failure, throws an exception or returns Y\_REBOOTCOUNTDOWN\_INVALID.

**module**→**get\_serialNumber()****YModule****module**→**serialNumber()****module.get\_serialNumber()**

Returns the serial number of the module, as set by the factory.

js	function <b>get_serialNumber</b> ( )
nodejs	function <b>get_serialNumber</b> ( )
php	function <b>get_serialNumber</b> ( )
cpp	string <b>get_serialNumber</b> ( )
m	-(NSString*) serialNumber
pas	function <b>get_serialNumber</b> ( ): string
vb	function <b>get_serialNumber</b> ( ) As String
cs	string <b>get_serialNumber</b> ( )
java	String <b>get_serialNumber</b> ( )
py	def <b>get_serialNumber</b> ( )
cmd	YModule <b>target</b> <b>get_serialNumber</b>

**Returns :**

a string corresponding to the serial number of the module, as set by the factory

On failure, throws an exception or returns Y\_SERIALNUMBER\_INVALID.

**module**→**get\_upTime()****YModule****module**→**upTime()****module.get\_upTime()**

Returns the number of milliseconds spent since the module was powered on.

js	function <b>get_upTime</b> ( )
nodejs	function <b>get_upTime</b> ( )
php	function <b>get_upTime</b> ( )
cpp	s64 <b>get_upTime</b> ( )
m	-(s64) upTime
pas	function <b>get_upTime</b> ( ): int64
vb	function <b>get_upTime</b> ( ) As Long
cs	long <b>get_upTime</b> ( )
java	long <b>get_upTime</b> ( )
py	def <b>get_upTime</b> ( )
cmd	YModule <b>target</b> <b>get_upTime</b>

**Returns :**

an integer corresponding to the number of milliseconds spent since the module was powered on

On failure, throws an exception or returns Y\_UPTIME\_INVALID.

**module**→**get\_usbBandwidth()****YModule****module**→**usbBandwidth()****module.get\_usbBandwidth()**

Returns the number of USB interfaces used by the module.

js	function <b>get_usbBandwidth</b> ( )
nodejs	function <b>get_usbBandwidth</b> ( )
php	function <b>get_usbBandwidth</b> ( )
cpp	Y_USBBANDWIDTH_enum <b>get_usbBandwidth</b> ( )
m	-(Y_USBBANDWIDTH_enum) usbBandwidth
pas	function <b>get_usbBandwidth</b> ( ): Integer
vb	function <b>get_usbBandwidth</b> ( ) As Integer
cs	int <b>get_usbBandwidth</b> ( )
java	int <b>get_usbBandwidth</b> ( )
py	def <b>get_usbBandwidth</b> ( )
cmd	YModule <b>target</b> <b>get_usbBandwidth</b>

**Returns :**

either Y\_USBBANDWIDTH\_SIMPLE or Y\_USBBANDWIDTH\_DOUBLE, according to the number of USB interfaces used by the module

On failure, throws an exception or returns Y\_USBBANDWIDTH\_INVALID.



**module**→**get\_usbCurrent()****YModule****module**→**usbCurrent()****module.get\_usbCurrent()**

Returns the current consumed by the module on the USB bus, in milli-amps.

js	function <b>get_usbCurrent</b> ( )
nodejs	function <b>get_usbCurrent</b> ( )
php	function <b>get_usbCurrent</b> ( )
cpp	int <b>get_usbCurrent</b> ( )
m	-(int) usbCurrent
pas	function <b>get_usbCurrent</b> ( ): LongInt
vb	function <b>get_usbCurrent</b> ( ) As Integer
cs	int <b>get_usbCurrent</b> ( )
java	int <b>get_usbCurrent</b> ( )
py	def <b>get_usbCurrent</b> ( )
cmd	YModule <b>target</b> <b>get_usbCurrent</b>

**Returns :**

an integer corresponding to the current consumed by the module on the USB bus, in milli-amps

On failure, throws an exception or returns Y\_USBCURRENT\_INVALID.

**module**→**get\_userData()****YModule****module**→**userData()****module.userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**module→isOnline()module.isOnline()****YModule**

Checks if the module is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there are valid cached values for the module, that have not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the requested module.

**Returns :**

`true` if the module can be reached, and `false` otherwise

**module**→**isOnline\_async()****YModule**

Checks if the module is currently reachable, without raising any error.

```
js function isOnline_async( callback, context)
nodejs function isOnline_async( callback, context)
```

If there are valid cached values for the module, that have not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the requested module.

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.

**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 module 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.

**module→load()module.load()****YModule**

Preloads the module cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

By default, whenever accessing a device, all module 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 traffic for instance.

**Parameters :**

**msValidity** an integer corresponding to the validity attributed to the loaded module parameters, in milliseconds

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**module→load\_async()****YModule**

Preloads the module 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 module 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 traffic for instance. 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 :**

- msValidity** an integer corresponding to the validity of the loaded module 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 module 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.

**module→nextModule()module.nextModule()****YModule**

Continues the module enumeration started using `yFirstModule()`.

js	function <b>nextModule</b> ( )
nodejs	function <b>nextModule</b> ( )
php	function <b>nextModule</b> ( )
cpp	YModule * <b>nextModule</b> ( )
m	-(YModule*) <b>nextModule</b>
pas	function <b>nextModule</b> ( ): TYModule
vb	function <b>nextModule</b> ( ) As YModule
cs	YModule <b>nextModule</b> ( )
java	YModule <b>nextModule</b> ( )
py	def <b>nextModule</b> ( )

**Returns :**

a pointer to a `YModule` object, corresponding to the next module found, or a `null` pointer if there are no more modules to enumerate.

**module→reboot()****module.reboot()****YModule**

Schedules a simple module reboot after the given number of seconds.

js	function <b>reboot</b> ( <b>secBeforeReboot</b> )
nodejs	function <b>reboot</b> ( <b>secBeforeReboot</b> )
php	function <b>reboot</b> ( <b>\$secBeforeReboot</b> )
cpp	int <b>reboot</b> ( int <b>secBeforeReboot</b> )
m	-(int) <b>reboot</b> : (int) <b>secBeforeReboot</b>
pas	function <b>reboot</b> ( <b>secBeforeReboot</b> : LongInt): LongInt
vb	function <b>reboot</b> ( ) As Integer
cs	int <b>reboot</b> ( int <b>secBeforeReboot</b> )
java	int <b>reboot</b> ( int <b>secBeforeReboot</b> )
py	def <b>reboot</b> ( <b>secBeforeReboot</b> )
cmd	YModule <b>target reboot secBeforeReboot</b>

**Parameters :**

**secBeforeReboot** number of seconds before rebooting

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.



---

**module**→**registerLogCallback()**  
**module.registerLogCallback()**

---

**YModule**

todo

cpp	void <b>registerLogCallback</b> ( YModuleLogCallback <b>callback</b> )
m	-(void) <b>registerLogCallback</b> : (YModuleLogCallback) <b>callback</b>
vb	function <b>registerLogCallback</b> ( ByVal <b>callback</b> As YModuleLogCallback) As Integer
cs	int <b>registerLogCallback</b> ( LogCallback <b>callback</b> )
py	def <b>registerLogCallback</b> ( <b>callback</b> )

**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.

**module→revertFromFlash()**  
**module.revertFromFlash()****YModule**

Reloads the settings stored in the nonvolatile memory, as when the module is powered on.

js	function <b>revertFromFlash</b> ( )
nodejs	function <b>revertFromFlash</b> ( )
php	function <b>revertFromFlash</b> ( )
cpp	int <b>revertFromFlash</b> ( )
m	-(int) <b>revertFromFlash</b>
pas	function <b>revertFromFlash</b> ( ): LongInt
vb	function <b>revertFromFlash</b> ( ) As Integer
cs	int <b>revertFromFlash</b> ( )
java	int <b>revertFromFlash</b> ( )
py	def <b>revertFromFlash</b> ( )
cmd	YModule <b>target</b> <b>revertFromFlash</b>

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**module→saveToFlash()****module.saveToFlash()****YModule**

Saves current settings in the nonvolatile memory of the module.

js	function <b>saveToFlash</b> ( )
nodejs	function <b>saveToFlash</b> ( )
php	function <b>saveToFlash</b> ( )
cpp	int <b>saveToFlash</b> ( )
m	-(int) <b>saveToFlash</b>
pas	function <b>saveToFlash</b> ( ): LongInt
vb	function <b>saveToFlash</b> ( ) As Integer
cs	int <b>saveToFlash</b> ( )
java	int <b>saveToFlash</b> ( )
py	def <b>saveToFlash</b> ( )
cmd	YModule <b>target</b> <b>saveToFlash</b>

Warning: the number of allowed save operations during a module life is limited (about 100000 cycles). Do not call this function within a loop.

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**module**→**set\_beacon()****YModule****module**→**setBeacon()****module.set\_beacon()**

Turns on or off the module localization beacon.

js	function <b>set_beacon</b> ( <b>newval</b> )
nodejs	function <b>set_beacon</b> ( <b>newval</b> )
php	function <b>set_beacon</b> ( <b>\$newval</b> )
cpp	int <b>set_beacon</b> ( Y_BEACON_enum <b>newval</b> )
m	-(int) setBeacon : (Y_BEACON_enum) <b>newval</b>
pas	function <b>set_beacon</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_beacon</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_beacon</b> ( int <b>newval</b> )
java	int <b>set_beacon</b> ( int <b>newval</b> )
py	def <b>set_beacon</b> ( <b>newval</b> )
cmd	YModule <b>target set_beacon newval</b>

**Parameters :**

**newval** either Y\_BEACON\_OFF or Y\_BEACON\_ON

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**module**→**set\_logicalName()****YModule****module**→**setLogicalName()****module.set\_logicalName()**

Changes the logical name of the module.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YModule <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 module

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**module**→**set\_luminosity()****YModule****module**→**setLuminosity()****module.set\_luminosity()**

Changes the luminosity of the module informative leds.

js	function <b>set_luminosity</b> ( <b>newval</b> )
nodejs	function <b>set_luminosity</b> ( <b>newval</b> )
php	function <b>set_luminosity</b> ( <b>\$newval</b> )
cpp	int <b>set_luminosity</b> ( int <b>newval</b> )
m	-(int) setLuminosity : (int) <b>newval</b>
pas	function <b>set_luminosity</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_luminosity</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_luminosity</b> ( int <b>newval</b> )
java	int <b>set_luminosity</b> ( int <b>newval</b> )
py	def <b>set_luminosity</b> ( <b>newval</b> )
cmd	YModule <b>target set_luminosity newval</b>

The parameter is a value between 0 and 100. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

**Parameters :**

**newval** an integer corresponding to the luminosity of the module informative leds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**module→set\_usbBandwidth()**  
**module→setUsbBandwidth()**  
**module.set\_usbBandwidth()**

**YModule**

Changes the number of USB interfaces used by the module.

js	function <b>set_usbBandwidth</b> ( <b>newval</b> )
nodejs	function <b>set_usbBandwidth</b> ( <b>newval</b> )
php	function <b>set_usbBandwidth</b> ( <b>\$newval</b> )
cpp	int <b>set_usbBandwidth</b> ( Y_USBBANDWIDTH_enum <b>newval</b> )
m	-(int) setUsbBandwidth : (Y_USBBANDWIDTH_enum) <b>newval</b>
pas	function <b>set_usbBandwidth</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_usbBandwidth</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_usbBandwidth</b> ( int <b>newval</b> )
java	int <b>set_usbBandwidth</b> ( int <b>newval</b> )
py	def <b>set_usbBandwidth</b> ( <b>newval</b> )
cmd	YModule <b>target</b> <b>set_usbBandwidth</b> <b>newval</b>

You must reboot the module after changing this setting.

#### Parameters :

**newval** either Y\_USBBANDWIDTH\_SIMPLE or Y\_USBBANDWIDTH\_DOUBLE, according to the number of USB interfaces used by the module

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**module**→**set\_userData()****YModule****module**→**setUserData()****module.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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



## module→triggerFirmwareUpdate() module.triggerFirmwareUpdate()

YModule

Schedules a module reboot into special firmware update mode.

js	function <b>triggerFirmwareUpdate</b> ( <b>secBeforeReboot</b> )
nodejs	function <b>triggerFirmwareUpdate</b> ( <b>secBeforeReboot</b> )
php	function <b>triggerFirmwareUpdate</b> ( <b>\$secBeforeReboot</b> )
cpp	int <b>triggerFirmwareUpdate</b> ( int <b>secBeforeReboot</b> )
m	-(int) <b>triggerFirmwareUpdate</b> : (int) <b>secBeforeReboot</b>
pas	function <b>triggerFirmwareUpdate</b> ( <b>secBeforeReboot</b> : LongInt): LongInt
vb	function <b>triggerFirmwareUpdate</b> ( ) As Integer
cs	int <b>triggerFirmwareUpdate</b> ( int <b>secBeforeReboot</b> )
java	int <b>triggerFirmwareUpdate</b> ( int <b>secBeforeReboot</b> )
py	def <b>triggerFirmwareUpdate</b> ( <b>secBeforeReboot</b> )
cmd	YModule <b>target triggerFirmwareUpdate</b> <b>secBeforeReboot</b>

### Parameters :

**secBeforeReboot** number of seconds before rebooting

### Returns :

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**module**→**wait\_async()****YModule**

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.

## 3.26. 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');
c++	#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)**

### 3. *Reference*

---

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YNetwork.FindNetwork()****YNetwork****yFindNetwork()YNetwork.FindNetwork()**

Retrieves a network interface for a given identifier.

js	function <b>yFindNetwork</b> ( <b>func</b> )
nodejs	function <b>FindNetwork</b> ( <b>func</b> )
php	function <b>yFindNetwork</b> ( <b>\$func</b> )
cpp	YNetwork* <b>yFindNetwork</b> ( const string& <b>func</b> )
m	YNetwork* <b>yFindNetwork</b> ( NSString* <b>func</b> )
pas	function <b>yFindNetwork</b> ( <b>func</b> : string): TYNetwork
vb	function <b>yFindNetwork</b> ( ByVal <b>func</b> As String) As YNetwork
cs	YNetwork <b>FindNetwork</b> ( string <b>func</b> )
java	YNetwork <b>FindNetwork</b> ( String <b>func</b> )
py	def <b>FindNetwork</b> ( <b>func</b> )

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()YNetwork.FirstNetwork()**

Starts the enumeration of network interfaces currently accessible.

js	function <b>yFirstNetwork</b> ( )
nodejs	function <b>FirstNetwork</b> ( )
php	function <b>yFirstNetwork</b> ( )
cpp	YNetwork* <b>yFirstNetwork</b> ( )
m	YNetwork* <b>yFirstNetwork</b> ( )
pas	function <b>yFirstNetwork</b> ( ): TYNetwork
vb	function <b>yFirstNetwork</b> ( ) As YNetwork
cs	YNetwork <b>FirstNetwork</b> ( )
java	YNetwork <b>FirstNetwork</b> ( )
py	def <b>FirstNetwork</b> ( )

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()network.callbackLogin()****YNetwork**

Connects to the notification callback and saves the credentials required to log into it.

js	function <b>callbackLogin</b> ( <b>username</b> , <b>password</b> )
nodejs	function <b>callbackLogin</b> ( <b>username</b> , <b>password</b> )
php	function <b>callbackLogin</b> ( <b>\$username</b> , <b>\$password</b> )
cpp	int <b>callbackLogin</b> ( string <b>username</b> , string <b>password</b> )
m	-(int) <b>callbackLogin</b> : (NSString*) <b>username</b> : (NSString*) <b>password</b>
pas	function <b>callbackLogin</b> ( <b>username</b> : string, <b>password</b> : string): integer
vb	function <b>callbackLogin</b> ( ByVal <b>username</b> As String, ByVal <b>password</b> As String) As Integer
cs	int <b>callbackLogin</b> ( string <b>username</b> , string <b>password</b> )
java	int <b>callbackLogin</b> ( String <b>username</b> , String <b>password</b> )
py	def <b>callbackLogin</b> ( <b>username</b> , <b>password</b> )
cmd	YNetwork <b>target callbackLogin</b> <b>username password</b>

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()****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 <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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()****network.get\_adminPassword()**

Returns a hash string if a password has been set for user "admin", or an empty string otherwise.

js	function <b>get_adminPassword</b> ( )
nodejs	function <b>get_adminPassword</b> ( )
php	function <b>get_adminPassword</b> ( )
cpp	string <b>get_adminPassword</b> ( )
m	-(NSString*) adminPassword
pas	function <b>get_adminPassword</b> ( ): string
vb	function <b>get_adminPassword</b> ( ) As String
cs	string <b>get_adminPassword</b> ( )
java	String <b>get_adminPassword</b> ( )
py	def <b>get_adminPassword</b> ( )
cmd	YNetwork <b>target</b> <b>get_adminPassword</b>

**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()****network**→**advertisedValue()****network.get\_advertisedValue()**

Returns the current value of the network interface (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YNetwork <b>target</b> <b>get_advertisedValue</b>

**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()****network.get\_callbackCredentials()**

Returns a hashed version of the notification callback credentials if set, or an empty string otherwise.

js	function <b>get_callbackCredentials</b> ( )
nodejs	function <b>get_callbackCredentials</b> ( )
php	function <b>get_callbackCredentials</b> ( )
cpp	string <b>get_callbackCredentials</b> ( )
m	-(NSString*) callbackCredentials
pas	function <b>get_callbackCredentials</b> ( ): string
vb	function <b>get_callbackCredentials</b> ( ) As String
cs	string <b>get_callbackCredentials</b> ( )
java	String <b>get_callbackCredentials</b> ( )
py	def <b>get_callbackCredentials</b> ( )
cmd	YNetwork <b>target</b> <b>get_callbackCredentials</b>

**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()****network.get\_callbackEncoding()**

Returns the encoding standard to use for representing notification values.

js	function <b>get_callbackEncoding</b> ( )
nodejs	function <b>get_callbackEncoding</b> ( )
php	function <b>get_callbackEncoding</b> ( )
cpp	Y_CALLBACKENCODING_enum <b>get_callbackEncoding</b> ( )
m	-(Y_CALLBACKENCODING_enum) callbackEncoding
pas	function <b>get_callbackEncoding</b> ( ): Integer
vb	function <b>get_callbackEncoding</b> ( ) As Integer
cs	int <b>get_callbackEncoding</b> ( )
java	int <b>get_callbackEncoding</b> ( )
py	def <b>get_callbackEncoding</b> ( )
cmd	YNetwork <b>target</b> <b>get_callbackEncoding</b>

**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()****network.get\_callbackMaxDelay()**

Returns the maximum waiting time between two callback notifications, in seconds.

js	function <b>get_callbackMaxDelay</b> ( )
nodejs	function <b>get_callbackMaxDelay</b> ( )
php	function <b>get_callbackMaxDelay</b> ( )
cpp	int <b>get_callbackMaxDelay</b> ( )
m	-(int) callbackMaxDelay
pas	function <b>get_callbackMaxDelay</b> ( ): LongInt
vb	function <b>get_callbackMaxDelay</b> ( ) As Integer
cs	int <b>get_callbackMaxDelay</b> ( )
java	int <b>get_callbackMaxDelay</b> ( )
py	def <b>get_callbackMaxDelay</b> ( )
cmd	YNetwork <b>target</b> <b>get_callbackMaxDelay</b>

**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()****network.get\_callbackMethod()**

Returns the HTTP method used to notify callbacks for significant state changes.

js	function <b>get_callbackMethod</b> ( )
nodejs	function <b>get_callbackMethod</b> ( )
php	function <b>get_callbackMethod</b> ( )
cpp	Y_CALLBACKMETHOD_enum <b>get_callbackMethod</b> ( )
m	-(Y_CALLBACKMETHOD_enum) callbackMethod
pas	function <b>get_callbackMethod</b> ( ): Integer
vb	function <b>get_callbackMethod</b> ( ) As Integer
cs	int <b>get_callbackMethod</b> ( )
java	int <b>get_callbackMethod</b> ( )
py	def <b>get_callbackMethod</b> ( )
cmd	YNetwork <b>target</b> <b>get_callbackMethod</b>

**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()****network.get\_callbackMinDelay()**

Returns the minimum waiting time between two callback notifications, in seconds.

js	function <b>get_callbackMinDelay</b> ( )
nodejs	function <b>get_callbackMinDelay</b> ( )
php	function <b>get_callbackMinDelay</b> ( )
cpp	int <b>get_callbackMinDelay</b> ( )
m	-(int) callbackMinDelay
pas	function <b>get_callbackMinDelay</b> ( ): LongInt
vb	function <b>get_callbackMinDelay</b> ( ) As Integer
cs	int <b>get_callbackMinDelay</b> ( )
java	int <b>get_callbackMinDelay</b> ( )
py	def <b>get_callbackMinDelay</b> ( )
cmd	YNetwork <b>target</b> <b>get_callbackMinDelay</b>

**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()****network.get\_callbackUrl()**

Returns the callback URL to notify of significant state changes.

js	function <b>get_callbackUrl</b> ( )
nodejs	function <b>get_callbackUrl</b> ( )
php	function <b>get_callbackUrl</b> ( )
cpp	string <b>get_callbackUrl</b> ( )
m	-(NSString*) callbackUrl
pas	function <b>get_callbackUrl</b> ( ): string
vb	function <b>get_callbackUrl</b> ( ) As String
cs	string <b>get_callbackUrl</b> ( )
java	String <b>get_callbackUrl</b> ( )
py	def <b>get_callbackUrl</b> ( )
cmd	YNetwork <b>target</b> <b>get_callbackUrl</b>

**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()****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).

js	function <b>get_discoverable</b> ( )
nodejs	function <b>get_discoverable</b> ( )
php	function <b>get_discoverable</b> ( )
cpp	Y_DISCOVERABLE_enum <b>get_discoverable</b> ( )
m	-(Y_DISCOVERABLE_enum) discoverable
pas	function <b>get_discoverable</b> ( ): Integer
vb	function <b>get_discoverable</b> ( ) As Integer
cs	int <b>get_discoverable</b> ( )
java	int <b>get_discoverable</b> ( )
py	def <b>get_discoverable</b> ( )
cmd	YNetwork <b>target</b> <b>get_discoverable</b>

**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()****network**→**errorMessage()****network.get\_errorMessage()**

Returns the error message of the latest error with the network interface.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the network interface object

**network**→**get\_errorType()****YNetwork****network**→**errorType()****network.get\_errorType()**

Returns the numerical error code of the latest error with the network interface.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the network interface object

**network**→**get\_friendlyName()****YNetwork****network**→**friendlyName()****network.get\_friendlyName()**

Returns a global identifier of the network interface in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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 example: `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()****network.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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()****network.get\_functionId()**

Returns the hardware identifier of the network interface, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

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()****network.get\_hardwareId()**

Returns the unique hardware identifier of the network interface in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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()****network.get\_ipAddress()**

Returns the IP address currently in use by the device.

js	function <b>get_ipAddress</b> ( )
nodejs	function <b>get_ipAddress</b> ( )
php	function <b>get_ipAddress</b> ( )
cpp	string <b>get_ipAddress</b> ( )
m	-(NSString*) ipAddress
pas	function <b>get_ipAddress</b> ( ): string
vb	function <b>get_ipAddress</b> ( ) As String
cs	string <b>get_ipAddress</b> ( )
java	String <b>get_ipAddress</b> ( )
py	def <b>get_ipAddress</b> ( )
cmd	YNetwork <b>target</b> <b>get_ipAddress</b>

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()****network.get\_logicalName()**

Returns the logical name of the network interface.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YNetwork <b>target</b> <b>get_logicalName</b>

**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()****network.get\_macAddress()**

Returns the MAC address of the network interface.

js	function <b>get_macAddress</b> ( )
nodejs	function <b>get_macAddress</b> ( )
php	function <b>get_macAddress</b> ( )
cpp	string <b>get_macAddress</b> ( )
m	-(NSString*) <b>macAddress</b>
pas	function <b>get_macAddress</b> ( ): string
vb	function <b>get_macAddress</b> ( ) As String
cs	string <b>get_macAddress</b> ( )
java	String <b>get_macAddress</b> ( )
py	def <b>get_macAddress</b> ( )
cmd	YNetwork <b>target</b> <b>get_macAddress</b>

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()network.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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()****network**→**module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**network→get\_poeCurrent()****YNetwork****network→poeCurrent()network.get\_poeCurrent()**

Returns the current consumed by the module from Power-over-Ethernet (PoE), in milli-amps.

js	function <b>get_poeCurrent</b> ( )
nodejs	function <b>get_poeCurrent</b> ( )
php	function <b>get_poeCurrent</b> ( )
cpp	int <b>get_poeCurrent</b> ( )
m	-(int) poeCurrent
pas	function <b>get_poeCurrent</b> ( ): LongInt
vb	function <b>get_poeCurrent</b> ( ) As Integer
cs	int <b>get_poeCurrent</b> ( )
java	int <b>get_poeCurrent</b> ( )
py	def <b>get_poeCurrent</b> ( )
cmd	YNetwork <b>target</b> <b>get_poeCurrent</b>

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()****network.get\_primaryDNS()**

Returns the IP address of the primary name server to be used by the module.

js	function <b>get_primaryDNS</b> ( )
nodejs	function <b>get_primaryDNS</b> ( )
php	function <b>get_primaryDNS</b> ( )
cpp	string <b>get_primaryDNS</b> ( )
m	-(NSString*) primaryDNS
pas	function <b>get_primaryDNS</b> ( ): string
vb	function <b>get_primaryDNS</b> ( ) As String
cs	string <b>get_primaryDNS</b> ( )
java	String <b>get_primaryDNS</b> ( )
py	def <b>get_primaryDNS</b> ( )
cmd	YNetwork <b>target</b> <b>get_primaryDNS</b>

**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()****network.get\_readiness()**

Returns the current established working mode of the network interface.

js	function <b>get_readiness</b> ( )
nodejs	function <b>get_readiness</b> ( )
php	function <b>get_readiness</b> ( )
cpp	Y_READINESS_enum <b>get_readiness</b> ( )
m	-(Y_READINESS_enum) readiness
pas	function <b>get_readiness</b> ( ): Integer
vb	function <b>get_readiness</b> ( ) As Integer
cs	int <b>get_readiness</b> ( )
java	int <b>get_readiness</b> ( )
py	def <b>get_readiness</b> ( )
cmd	YNetwork <b>target</b> <b>get_readiness</b>

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()****network.get\_router()**

Returns the IP address of the router on the device subnet (default gateway).

js	function <b>get_router</b> ( )
nodejs	function <b>get_router</b> ( )
php	function <b>get_router</b> ( )
cpp	string <b>get_router</b> ( )
m	-(NSString*) router
pas	function <b>get_router</b> ( ): string
vb	function <b>get_router</b> ( ) As String
cs	string <b>get_router</b> ( )
java	String <b>get_router</b> ( )
py	def <b>get_router</b> ( )
cmd	YNetwork <b>target</b> <b>get_router</b>

**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()****network.get\_secondaryDNS()**

Returns the IP address of the secondary name server to be used by the module.

js	function <b>get_secondaryDNS</b> ( )
nodejs	function <b>get_secondaryDNS</b> ( )
php	function <b>get_secondaryDNS</b> ( )
cpp	string <b>get_secondaryDNS</b> ( )
m	-(NSString*) secondaryDNS
pas	function <b>get_secondaryDNS</b> ( ): string
vb	function <b>get_secondaryDNS</b> ( ) As String
cs	string <b>get_secondaryDNS</b> ( )
java	String <b>get_secondaryDNS</b> ( )
py	def <b>get_secondaryDNS</b> ( )
cmd	YNetwork <b>target</b> <b>get_secondaryDNS</b>

**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()****network.get\_subnetMask()**

Returns the subnet mask currently used by the device.

js	function <b>get_subnetMask</b> ( )
nodejs	function <b>get_subnetMask</b> ( )
php	function <b>get_subnetMask</b> ( )
cpp	string <b>get_subnetMask</b> ( )
m	-(NSString*) subnetMask
pas	function <b>get_subnetMask</b> ( ): string
vb	function <b>get_subnetMask</b> ( ) As String
cs	string <b>get_subnetMask</b> ( )
java	String <b>get_subnetMask</b> ( )
py	def <b>get_subnetMask</b> ( )
cmd	YNetwork <b>target</b> <b>get_subnetMask</b>

**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()****network.userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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()****network→userPassword()****network.get\_userPassword()**

Returns a hash string if a password has been set for "user" user, or an empty string otherwise.

js	function <b>get_userPassword</b> ( )
nodejs	function <b>get_userPassword</b> ( )
php	function <b>get_userPassword</b> ( )
cpp	string <b>get_userPassword</b> ( )
m	-(NSString*) userPassword
pas	function <b>get_userPassword</b> ( ): string
vb	function <b>get_userPassword</b> ( ) As String
cs	string <b>get_userPassword</b> ( )
java	String <b>get_userPassword</b> ( )
py	def <b>get_userPassword</b> ( )
cmd	YNetwork <b>target</b> <b>get_userPassword</b>

**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()****network.get\_wwwWatchdogDelay()**

Returns the allowed downtime of the WWW link (in seconds) before triggering an automated reboot to try to recover Internet connectivity.

js	function <b>get_wwwWatchdogDelay</b> ( )
nodejs	function <b>get_wwwWatchdogDelay</b> ( )
php	function <b>get_wwwWatchdogDelay</b> ( )
cpp	int <b>get_wwwWatchdogDelay</b> ( )
m	-(int) <b>wwwWatchdogDelay</b>
pas	function <b>get_wwwWatchdogDelay</b> ( ): LongInt
vb	function <b>get_wwwWatchdogDelay</b> ( ) As Integer
cs	int <b>get_wwwWatchdogDelay</b> ( )
java	int <b>get_wwwWatchdogDelay</b> ( )
py	def <b>get_wwwWatchdogDelay</b> ( )
cmd	YNetwork <b>target</b> <b>get_wwwWatchdogDelay</b>

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()network.isOnline()****YNetwork**

Checks if the network interface is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()network.load()****YNetwork**

Preloads the network interface cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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 traffic 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()****network.nextNetwork()****YNetwork**

Continues the enumeration of network interfaces started using `yFirstNetwork()`.

js	function <b>nextNetwork</b> ( )
nodejs	function <b>nextNetwork</b> ( )
php	function <b>nextNetwork</b> ( )
cpp	YNetwork * <b>nextNetwork</b> ( )
m	-(YNetwork*) <b>nextNetwork</b>
pas	function <b>nextNetwork</b> ( ): TYNetwork
vb	function <b>nextNetwork</b> ( ) As YNetwork
cs	YNetwork <b>nextNetwork</b> ( )
java	YNetwork <b>nextNetwork</b> ( )
py	def <b>nextNetwork</b> ( )

**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()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() network.registerValueCallback()

YNetwork

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YNetworkValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YNetworkValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYNetworkValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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()****network.set\_adminPassword()**

Changes the password for the "admin" user.

js	function <b>set_adminPassword</b> ( <b>newval</b> )
nodejs	function <b>set_adminPassword</b> ( <b>newval</b> )
php	function <b>set_adminPassword</b> ( <b>\$newval</b> )
cpp	int <b>set_adminPassword</b> ( const string& <b>newval</b> )
m	-(int) setAdminPassword : (NSString*) <b>newval</b>
pas	function <b>set_adminPassword</b> ( <b>newval</b> : string): integer
vb	function <b>set_adminPassword</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_adminPassword</b> ( string <b>newval</b> )
java	int <b>set_adminPassword</b> ( String <b>newval</b> )
py	def <b>set_adminPassword</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_adminPassword</b> <b>newval</b>

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()****network→setCallbackCredentials()****network.set\_callbackCredentials()**

Changes the credentials required to connect to the callback address.

js	function <b>set_callbackCredentials</b> ( <b>newval</b> )
nodejs	function <b>set_callbackCredentials</b> ( <b>newval</b> )
php	function <b>set_callbackCredentials</b> ( <b>\$newval</b> )
c++	int <b>set_callbackCredentials</b> ( const string& <b>newval</b> )
m	-(int) setCallbackCredentials : (NSString*) <b>newval</b>
pas	function <b>set_callbackCredentials</b> ( <b>newval</b> : string): integer
vb	function <b>set_callbackCredentials</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_callbackCredentials</b> ( string <b>newval</b> )
java	int <b>set_callbackCredentials</b> ( String <b>newval</b> )
py	def <b>set_callbackCredentials</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_callbackCredentials</b> <b>newval</b>

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 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()****network.set\_callbackEncoding()**

Changes the encoding standard to use for representing notification values.

js	function <b>set_callbackEncoding</b> ( <b>newval</b> )
nodejs	function <b>set_callbackEncoding</b> ( <b>newval</b> )
php	function <b>set_callbackEncoding</b> ( <b>\$newval</b> )
cpp	int <b>set_callbackEncoding</b> ( Y_CALLBACKENCODING_enum <b>newval</b> )
m	-(int) setCallbackEncoding : (Y_CALLBACKENCODING_enum) <b>newval</b>
pas	function <b>set_callbackEncoding</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_callbackEncoding</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_callbackEncoding</b> ( int <b>newval</b> )
java	int <b>set_callbackEncoding</b> ( int <b>newval</b> )
py	def <b>set_callbackEncoding</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_callbackEncoding</b> <b>newval</b>

**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()****network**→**setCallbackMaxDelay()****network.set\_callbackMaxDelay()**

Changes the maximum waiting time between two callback notifications, in seconds.

js	function <b>set_callbackMaxDelay</b> ( <b>newval</b> )
nodejs	function <b>set_callbackMaxDelay</b> ( <b>newval</b> )
php	function <b>set_callbackMaxDelay</b> ( <b>\$newval</b> )
cpp	int <b>set_callbackMaxDelay</b> ( int <b>newval</b> )
m	-(int) setCallbackMaxDelay : (int) <b>newval</b>
pas	function <b>set_callbackMaxDelay</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_callbackMaxDelay</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_callbackMaxDelay</b> ( int <b>newval</b> )
java	int <b>set_callbackMaxDelay</b> ( int <b>newval</b> )
py	def <b>set_callbackMaxDelay</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_callbackMaxDelay</b> <b>newval</b>

**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()****network.set\_callbackMethod()**

Changes the HTTP method used to notify callbacks for significant state changes.

js	function <b>set_callbackMethod</b> ( <b>newval</b> )
nodejs	function <b>set_callbackMethod</b> ( <b>newval</b> )
php	function <b>set_callbackMethod</b> ( <b>\$newval</b> )
cpp	int <b>set_callbackMethod</b> ( Y_CALLBACKMETHOD_enum <b>newval</b> )
m	-(int) setCallbackMethod : (Y_CALLBACKMETHOD_enum) <b>newval</b>
pas	function <b>set_callbackMethod</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_callbackMethod</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_callbackMethod</b> ( int <b>newval</b> )
java	int <b>set_callbackMethod</b> ( int <b>newval</b> )
py	def <b>set_callbackMethod</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_callbackMethod</b> <b>newval</b>

**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()****network.set\_callbackMinDelay()**

Changes the minimum waiting time between two callback notifications, in seconds.

js	function <b>set_callbackMinDelay</b> ( <b>newval</b> )
nodejs	function <b>set_callbackMinDelay</b> ( <b>newval</b> )
php	function <b>set_callbackMinDelay</b> ( <b>\$newval</b> )
cpp	int <b>set_callbackMinDelay</b> ( int <b>newval</b> )
m	-(int) setCallbackMinDelay : (int) <b>newval</b>
pas	function <b>set_callbackMinDelay</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_callbackMinDelay</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_callbackMinDelay</b> ( int <b>newval</b> )
java	int <b>set_callbackMinDelay</b> ( int <b>newval</b> )
py	def <b>set_callbackMinDelay</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_callbackMinDelay</b> <b>newval</b>

**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()****network.set\_callbackUrl()**

Changes the callback URL to notify significant state changes.

js	function <b>set_callbackUrl</b> ( <b>newval</b> )
nodejs	function <b>set_callbackUrl</b> ( <b>newval</b> )
php	function <b>set_callbackUrl</b> ( <b>\$newval</b> )
cpp	int <b>set_callbackUrl</b> ( const string& <b>newval</b> )
m	-(int) setCallbackUrl : (NSString*) <b>newval</b>
pas	function <b>set_callbackUrl</b> ( <b>newval</b> : string): integer
vb	function <b>set_callbackUrl</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_callbackUrl</b> ( string <b>newval</b> )
java	int <b>set_callbackUrl</b> ( String <b>newval</b> )
py	def <b>set_callbackUrl</b> ( <b>newval</b> )
cmd	YNetwork <b>target set_callbackUrl newval</b>

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()****network**→**setDiscoverable()****network.set\_discoverable()**

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 <b>set_discoverable</b> ( <b>newval</b> )
nodejs	function <b>set_discoverable</b> ( <b>newval</b> )
php	function <b>set_discoverable</b> ( <b>\$newval</b> )
cpp	int <b>set_discoverable</b> ( Y_DISCOVERABLE_enum <b>newval</b> )
m	-(int) setDiscoverable : (Y_DISCOVERABLE_enum) <b>newval</b>
pas	function <b>set_discoverable</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_discoverable</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_discoverable</b> ( int <b>newval</b> )
java	int <b>set_discoverable</b> ( int <b>newval</b> )
py	def <b>set_discoverable</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_discoverable</b> <b>newval</b>

**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()****network.set\_logicalName()**

Changes the logical name of the network interface.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_logicalName</b> <b>newval</b>

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()network.set\_primaryDNS()**

Changes the IP address of the primary name server to be used by the module.

js	function <b>set_primaryDNS</b> ( <b>newval</b> )
nodejs	function <b>set_primaryDNS</b> ( <b>newval</b> )
php	function <b>set_primaryDNS</b> ( <b>\$newval</b> )
cpp	int <b>set_primaryDNS</b> ( const string& <b>newval</b> )
m	-(int) setPrimaryDNS : (NSString*) <b>newval</b>
pas	function <b>set_primaryDNS</b> ( <b>newval</b> : string): integer
vb	function <b>set_primaryDNS</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_primaryDNS</b> ( string <b>newval</b> )
java	int <b>set_primaryDNS</b> ( String <b>newval</b> )
py	def <b>set_primaryDNS</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_primaryDNS</b> <b>newval</b>

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()****network.set\_secondaryDNS()**

Changes the IP address of the secondary name server to be used by the module.

js	function <b>set_secondaryDNS</b> ( <b>newval</b> )
nodejs	function <b>set_secondaryDNS</b> ( <b>newval</b> )
php	function <b>set_secondaryDNS</b> ( <b>\$newval</b> )
cpp	int <b>set_secondaryDNS</b> ( const string& <b>newval</b> )
m	-(int) setSecondaryDNS : (NSString*) <b>newval</b>
pas	function <b>set_secondaryDNS</b> ( <b>newval</b> : string): integer
vb	function <b>set_secondaryDNS</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_secondaryDNS</b> ( string <b>newval</b> )
java	int <b>set_secondaryDNS</b> ( String <b>newval</b> )
py	def <b>set_secondaryDNS</b> ( <b>newval</b> )
cmd	YNetwork <b>target set_secondaryDNS newval</b>

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()****network**→**setUserData()****network.set\_userdata()**

Stores a user context provided as argument in the `userData` attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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()**  
**network→setUserPassword()**  
**network.set\_userPassword()**

**YNetwork**

Changes the password for the "user" user.

js	function <b>set_userPassword</b> ( <b>newval</b> )
nodejs	function <b>set_userPassword</b> ( <b>newval</b> )
php	function <b>set_userPassword</b> ( <b>\$newval</b> )
cpp	int <b>set_userPassword</b> ( const string& <b>newval</b> )
m	-(int) setUserPassword : (NSString*) <b>newval</b>
pas	function <b>set_userPassword</b> ( <b>newval</b> : string): integer
vb	function <b>set_userPassword</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_userPassword</b> ( string <b>newval</b> )
java	int <b>set_userPassword</b> ( String <b>newval</b> )
py	def <b>set_userPassword</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_userPassword</b> <b>newval</b>

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()****network.set\_wwwWatchdogDelay()**

Changes the allowed downtime of the WWW link (in seconds) before triggering an automated reboot to try to recover Internet connectivity.

js	function <b>set_wwwWatchdogDelay</b> ( <b>newval</b> )
nodejs	function <b>set_wwwWatchdogDelay</b> ( <b>newval</b> )
php	function <b>set_wwwWatchdogDelay</b> ( <b>\$newval</b> )
c++	int <b>set_wwwWatchdogDelay</b> ( int <b>newval</b> )
m	-(int) setWwwWatchdogDelay : (int) <b>newval</b>
pas	function <b>set_wwwWatchdogDelay</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_wwwWatchdogDelay</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_wwwWatchdogDelay</b> ( int <b>newval</b> )
java	int <b>set_wwwWatchdogDelay</b> ( int <b>newval</b> )
py	def <b>set_wwwWatchdogDelay</b> ( <b>newval</b> )
cmd	YNetwork <b>target</b> <b>set_wwwWatchdogDelay</b> <b>newval</b>

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()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 :**

<b>fallbackIpAddr</b>	fallback IP address, to be used when no DHCP reply is received
<b>fallbackSubnetMaskLen</b>	fallback subnet mask length when no DHCP reply is received, as an integer (eg. 24 means 255.255.255.0)
<b>fallbackRouter</b>	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()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.

## 3.27. OS control

The OScontrol object allows some control over the operating system running a VirtualHub. OsControl is available on the VirtualHub software only. This feature must be activated at the VirtualHub start up with -o option.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_oscontrol.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YOsControl = yoctolib.YOsControl;
php	require_once('yocto_oscontrol.php');
c++	#include "yocto_oscontrol.h"
m	#import "yocto_oscontrol.h"
pas	uses yocto_oscontrol;
vb	yocto_oscontrol.vb
cs	yocto_oscontrol.cs
java	import com.yoctopuce.YoctoAPI.YOsControl;
py	from yocto_oscontrol import *

### Global functions

#### yFindOsControl(func)

Retrieves OS control for a given identifier.

#### yFirstOsControl()

Starts the enumeration of OS control currently accessible.

### YOsControl methods

#### oscontrol→describe()

Returns a short text that describes unambiguously the instance of the OS control in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### oscontrol→get\_advertisedValue()

Returns the current value of the OS control (no more than 6 characters).

#### oscontrol→get\_errorMessage()

Returns the error message of the latest error with the OS control.

#### oscontrol→get\_errorType()

Returns the numerical error code of the latest error with the OS control.

#### oscontrol→get\_friendlyName()

Returns a global identifier of the OS control in the format MODULE\_NAME . FUNCTION\_NAME.

#### oscontrol→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### oscontrol→get\_functionId()

Returns the hardware identifier of the OS control, without reference to the module.

#### oscontrol→get\_hardwareId()

Returns the unique hardware identifier of the OS control in the form SERIAL . FUNCTIONID.

#### oscontrol→get\_logicalName()

Returns the logical name of the OS control.

#### oscontrol→get\_module()

Gets the YModule object for the device on which the function is located.

#### oscontrol→get\_module\_async(callback, context)



Gets the `YModule` object for the device on which the function is located (asynchronous version).

**`oscontrol→get_shutdownCountdown()`**

Returns the remaining number of seconds before the OS shutdown, or zero when no shutdown has been scheduled.

**`oscontrol→get_userData()`**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

**`oscontrol→isOnline()`**

Checks if the OS control is currently reachable, without raising any error.

**`oscontrol→isOnline_async(callback, context)`**

Checks if the OS control is currently reachable, without raising any error (asynchronous version).

**`oscontrol→load(msValidity)`**

Preloads the OS control cache with a specified validity duration.

**`oscontrol→load_async(msValidity, callback, context)`**

Preloads the OS control cache with a specified validity duration (asynchronous version).

**`oscontrol→nextOsControl()`**

Continues the enumeration of OS control started using `yFirstOsControl()`.

**`oscontrol→registerValueCallback(callback)`**

Registers the callback function that is invoked on every change of advertised value.

**`oscontrol→set_logicalName(newval)`**

Changes the logical name of the OS control.

**`oscontrol→set_userData(data)`**

Stores a user context provided as argument in the `userData` attribute of the function.

**`oscontrol→shutdown(secBeforeShutDown)`**

Schedules an OS shutdown after a given number of seconds.

**`oscontrol→wait_async(callback, context)`**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YOsControl.FindOsControl()****YOsControl****yFindOsControl()YOsControl.FindOsControl()**

Retrieves OS control for a given identifier.

js	function <b>yFindOsControl</b> ( <b>func</b> )
nodejs	function <b>FindOsControl</b> ( <b>func</b> )
php	function <b>yFindOsControl</b> ( <b>\$func</b> )
cpp	YOsControl* <b>yFindOsControl</b> ( const string& <b>func</b> )
m	YOsControl* <b>yFindOsControl</b> ( NSString* <b>func</b> )
pas	function <b>yFindOsControl</b> ( <b>func</b> : string): TYOsControl
vb	function <b>yFindOsControl</b> ( ByVal <b>func</b> As String) As YOsControl
cs	YOsControl <b>FindOsControl</b> ( string <b>func</b> )
java	YOsControl <b>FindOsControl</b> ( String <b>func</b> )
py	def <b>FindOsControl</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the OS control is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YOsControl.isOnline()` to test if the OS control is indeed online at a given time. In case of ambiguity when looking for OS control 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 OS control

**Returns :**

a `YOsControl` object allowing you to drive the OS control.

**YOsControl.FirstOsControl()****YOsControl****yFirstOsControl()YOsControl.FirstOsControl()**

Starts the enumeration of OS control currently accessible.

js	function <b>yFirstOsControl</b> ( )
nodejs	function <b>FirstOsControl</b> ( )
php	function <b>yFirstOsControl</b> ( )
cpp	YOsControl* <b>yFirstOsControl</b> ( )
m	YOsControl* <b>yFirstOsControl</b> ( )
pas	function <b>yFirstOsControl</b> ( ): TYOsControl
vb	function <b>yFirstOsControl</b> ( ) As YOsControl
cs	YOsControl <b>FirstOsControl</b> ( )
java	YOsControl <b>FirstOsControl</b> ( )
py	def <b>FirstOsControl</b> ( )

Use the method `YOsControl.nextOsControl( )` to iterate on next OS control.

**Returns :**

a pointer to a `YOsControl` object, corresponding to the first OS control currently online, or a `null` pointer if there are none.

**oscontrol→describe()****oscontrol.describe()****YOsControl**

Returns a short text that describes unambiguously the instance of the OS control in the form  
 TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 OS control (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**oscontrol→get\_advertisedValue()****YOsControl****oscontrol→advertisedValue()****oscontrol.get\_advertisedValue()**

Returns the current value of the OS control (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YOsControl <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the OS control (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**oscontrol→get\_errorMessage()****YOsControl****oscontrol→errorMessage()****oscontrol.errorMessage()**

Returns the error message of the latest error with the OS control.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the OS control object

**oscontrol**→**get\_errorType()****YOsControl****oscontrol**→**errorType()****oscontrol.get\_errorType()**

Returns the numerical error code of the latest error with the OS control.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the OS control object

**oscontrol→get\_friendlyName()****YOsControl****oscontrol→friendlyName()****oscontrol.get\_friendlyName()**

Returns a global identifier of the OS control in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the OS control if they are defined, otherwise the serial number of the module and the hardware identifier of the OS control (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the OS control using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.



**oscontrol**→**get\_functionDescriptor()**  
**oscontrol**→**functionDescriptor()**  
**oscontrol.get\_functionDescriptor()**

**YOsControl**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**oscontrol**→**get\_functionId()****YOsControl****oscontrol**→**functionId()****oscontrol.get\_functionId()**

Returns the hardware identifier of the OS control, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the OS control (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**oscontrol→get\_hardwareId()****YOsControl****oscontrol→hardwareId()oscontrol.get\_hardwareId()**

Returns the unique hardware identifier of the OS control in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the OS control. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the OS control (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**oscontrol→get\_logicalName()****YOsControl****oscontrol→logicalName()****oscontrol.get\_logicalName()**

Returns the logical name of the OS control.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YOsControl <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the OS control. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**oscontrol→get\_module()****YOsControl****oscontrol→module()oscontrol.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**oscontrol**→**get\_module\_async()****YOsControl****oscontrol**→**module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**oscontrol→get\_shutdownCountdown()****YOsControl****oscontrol→shutdownCountdown()****oscontrol.get\_shutdownCountdown()**

Returns the remaining number of seconds before the OS shutdown, or zero when no shutdown has been scheduled.

js	function <b>get_shutdownCountdown</b> ( )
nodejs	function <b>get_shutdownCountdown</b> ( )
php	function <b>get_shutdownCountdown</b> ( )
cpp	int <b>get_shutdownCountdown</b> ( )
m	-(int) shutdownCountdown
pas	function <b>get_shutdownCountdown</b> ( ): LongInt
vb	function <b>get_shutdownCountdown</b> ( ) As Integer
cs	int <b>get_shutdownCountdown</b> ( )
java	int <b>get_shutdownCountdown</b> ( )
py	def <b>get_shutdownCountdown</b> ( )
cmd	YOsControl <b>target</b> <b>get_shutdownCountdown</b>

**Returns :**

an integer corresponding to the remaining number of seconds before the OS shutdown, or zero when no shutdown has been scheduled

On failure, throws an exception or returns Y\_SHUTDOWNCOUNTDOWN\_INVALID.

**oscontrol**→**get\_userData()****YOsControl****oscontrol**→**userData()****oscontrol.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.



**oscontrol→isOnline()oscontrol.isOnline()****YOsControl**

Checks if the OS control is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the OS control 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 OS control.

**Returns :**

`true` if the OS control can be reached, and `false` otherwise

Checks if the OS control 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 OS control 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.

**oscontrol→load()oscontrol.load()****YOsControl**

Preloads the OS control cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

Preloads the OS control 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 traffic 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.

## oscontrol→nextOsControl() oscontrol.nextOsControl()

YOsControl

Continues the enumeration of OS control started using `yFirstOsControl()`.

js	function <b>nextOsControl</b> ( )
nodejs	function <b>nextOsControl</b> ( )
php	function <b>nextOsControl</b> ( )
cpp	YOsControl * <b>nextOsControl</b> ( )
m	-(YOsControl*) <b>nextOsControl</b>
pas	function <b>nextOsControl</b> ( ): TYOsControl
vb	function <b>nextOsControl</b> ( ) As YOsControl
cs	YOsControl <b>nextOsControl</b> ( )
java	YOsControl <b>nextOsControl</b> ( )
py	def <b>nextOsControl</b> ( )

### Returns :

a pointer to a `YOsControl` object, corresponding to OS control currently online, or a `null` pointer if there are no more OS control to enumerate.

## oscontrol→registerValueCallback() oscontrol.registerValueCallback()

YOsControl

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YOsControlValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YOsControlValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYOsControlValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**oscontrol→set\_logicalName()****YOsControl****oscontrol→setLogicalName()****oscontrol.set\_logicalName()**

Changes the logical name of the OS control.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YOsControl <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 OS control.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**oscontrol**→**set\_userdata()****YOsControl****oscontrol**→**setUserData()****oscontrol.set\_userdata()**

Stores a user context provided as argument in the `userData` attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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



**oscontrol→shutdown()****oscontrol.shutdown()****YOsControl**

Schedules an OS shutdown after a given number of seconds.

js	function <b>shutdown</b> ( <b>secBeforeShutDown</b> )
nodejs	function <b>shutdown</b> ( <b>secBeforeShutDown</b> )
php	function <b>shutdown</b> ( <b>\$secBeforeShutDown</b> )
cpp	int <b>shutdown</b> ( int <b>secBeforeShutDown</b> )
m	-(int) <b>shutdown</b> : (int) <b>secBeforeShutDown</b>
pas	function <b>shutdown</b> ( <b>secBeforeShutDown</b> : LongInt): LongInt
vb	function <b>shutdown</b> ( ) As Integer
cs	int <b>shutdown</b> ( int <b>secBeforeShutDown</b> )
java	int <b>shutdown</b> ( int <b>secBeforeShutDown</b> )
py	def <b>shutdown</b> ( <b>secBeforeShutDown</b> )
cmd	YOsControl <b>target shutdown secBeforeShutDown</b>

**Parameters :**

**secBeforeShutDown** number of seconds before shutdown

**Returns :**

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**oscontrol→wait\_async()****YOsControl**

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.

## 3.28. Power function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_power.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPower = yoctolib.YPower;
php	require_once('yocto_power.php');
c++	#include "yocto_power.h"
m	#import "yocto_power.h"
pas	uses yocto_power;
vb	yocto_power.vb
cs	yocto_power.cs
java	import com.yoctopuce.YoctoAPI.YPower;
py	from yocto_power import *

### Global functions

#### yFindPower(func)

Retrieves a electrical power sensor for a given identifier.

#### yFirstPower()

Starts the enumeration of electrical power sensors currently accessible.

### YPower methods

#### power→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### power→describe()

Returns a short text that describes unambiguously the instance of the electrical power sensor in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### power→get\_advertisedValue()

Returns the current value of the electrical power sensor (no more than 6 characters).

#### power→get\_cosPhi()

Returns the power factor (the ratio between the real power consumed, measured in W, and the apparent power provided, measured in VA).

#### power→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### power→get\_currentValue()

Returns the current measure for the electrical power.

#### power→get\_errorMessage()

Returns the error message of the latest error with the electrical power sensor.

#### power→get\_errorType()

Returns the numerical error code of the latest error with the electrical power sensor.

#### power→get\_friendlyName()

Returns a global identifier of the electrical power sensor in the format MODULE\_NAME . FUNCTION\_NAME.

#### power→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### power→get\_functionId()

Returns the hardware identifier of the electrical power sensor, without reference to the module.

**power**→**get\_hardwareId()**

Returns the unique hardware identifier of the electrical power sensor in the form `SERIAL.FUNCTIONID`.

**power**→**get\_highestValue()**

Returns the maximal value observed for the electrical power.

**power**→**get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**power**→**get\_logicalName()**

Returns the logical name of the electrical power sensor.

**power**→**get\_lowestValue()**

Returns the minimal value observed for the electrical power.

**power**→**get\_meter()**

Returns the energy counter, maintained by the wattmeter by integrating the power consumption over time.

**power**→**get\_meterTimer()**

Returns the elapsed time since last energy counter reset, in seconds.

**power**→**get\_module()**

Gets the `YModule` object for the device on which the function is located.

**power**→**get\_module\_async(callback, context)**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

**power**→**get\_recordedData(startTime, endTime)**

Retrieves a `DataSet` object holding historical data for this sensor, for a specified time interval.

**power**→**get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**power**→**get\_resolution()**

Returns the resolution of the measured values.

**power**→**get\_unit()**

Returns the measuring unit for the electrical power.

**power**→**get\_userData()**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

**power**→**isOnline()**

Checks if the electrical power sensor is currently reachable, without raising any error.

**power**→**isOnline\_async(callback, context)**

Checks if the electrical power sensor is currently reachable, without raising any error (asynchronous version).

**power**→**load(msValidity)**

Preloads the electrical power sensor cache with a specified validity duration.

**power**→**loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

**power**→**load\_async(msValidity, callback, context)**

Preloads the electrical power sensor cache with a specified validity duration (asynchronous version).

**power**→**nextPower()**

Continues the enumeration of electrical power sensors started using `yFirstPower()`.

**power**→**registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**power**→**registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**power→reset()**

Resets the energy counter.

**power→set\_highestValue(newval)**

Changes the recorded maximal value observed pour the electrical power.

**power→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**power→set\_logicalName(newval)**

Changes the logical name of the electrical power sensor.

**power→set\_lowestValue(newval)**

Changes the recorded minimal value observed pour the electrical power.

**power→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**power→set\_resolution(newval)**

Changes the resolution of the measured values.

**power→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**power→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YPower.FindPower()****YPower****yFindPower()YPower.FindPower()**

Retrieves a electrical power sensor for a given identifier.

js	function <b>yFindPower</b> ( <b>func</b> )
nodejs	function <b>FindPower</b> ( <b>func</b> )
php	function <b>yFindPower</b> ( <b>\$func</b> )
cpp	YPower* <b>yFindPower</b> ( const string& <b>func</b> )
m	YPower* <b>yFindPower</b> ( NSString* <b>func</b> )
pas	function <b>yFindPower</b> ( <b>func</b> : string): TYPower
vb	function <b>yFindPower</b> ( ByVal <b>func</b> As String) As YPower
cs	YPower <b>FindPower</b> ( string <b>func</b> )
java	YPower <b>FindPower</b> ( String <b>func</b> )
py	def <b>FindPower</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the electrical power sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YPower.isOnline()` to test if the electrical power sensor is indeed online at a given time. In case of ambiguity when looking for a electrical power sensor 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 electrical power sensor

**Returns :**

a `YPower` object allowing you to drive the electrical power sensor.

## YPower.FirstPower() yFirstPower()YPower.FirstPower()

YPower

Starts the enumeration of electrical power sensors currently accessible.

js	function <b>yFirstPower</b> ( )
nodejs	function <b>FirstPower</b> ( )
php	function <b>yFirstPower</b> ( )
cpp	YPower* <b>yFirstPower</b> ( )
m	YPower* <b>yFirstPower</b> ( )
pas	function <b>yFirstPower</b> ( ): TYPower
vb	function <b>yFirstPower</b> ( ) As YPower
cs	YPower <b>FirstPower</b> ( )
java	YPower <b>FirstPower</b> ( )
py	def <b>FirstPower</b> ( )

Use the method `YPower.nextPower( )` to iterate on next electrical power sensors.

### Returns :

a pointer to a `YPower` object, corresponding to the first electrical power sensor currently online, or a `null` pointer if there are none.

## power→calibrateFromPoints() power.calibrateFromPoints()

YPower

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YPower target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**power→describe()power.describe()****YPower**

Returns a short text that describes unambiguously the instance of the electrical power sensor in the form `TYPE (NAME) = SERIAL . FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 electrical power sensor (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**power**→**get\_advertisedValue()****YPower****power**→**advertisedValue()****power.get\_advertisedValue()**

Returns the current value of the electrical power sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YPower <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the electrical power sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**power**→**get\_cosPhi()****YPower****power**→**cosPhi()****power.get\_cosPhi()**

Returns the power factor (the ratio between the real power consumed, measured in W, and the apparent power provided, measured in VA).

js	function <b>get_cosPhi</b> ( )
nodejs	function <b>get_cosPhi</b> ( )
php	function <b>get_cosPhi</b> ( )
cpp	double <b>get_cosPhi</b> ( )
m	-(double) cosPhi
pas	function <b>get_cosPhi</b> ( ): double
vb	function <b>get_cosPhi</b> ( ) As Double
cs	double <b>get_cosPhi</b> ( )
java	double <b>get_cosPhi</b> ( )
py	def <b>get_cosPhi</b> ( )
cmd	YPower <b>target</b> <b>get_cosPhi</b>

**Returns :**

a floating point number corresponding to the power factor (the ratio between the real power consumed, measured in W, and the apparent power provided, measured in VA)

On failure, throws an exception or returns Y\_COSPHI\_INVALID.

**power**→**get\_currentRawValue()****YPower****power**→**currentRawValue()****power.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YPower <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**power**→**get\_currentValue()****YPower****power**→**currentValue()****power.get\_currentValue()**

Returns the current measure for the electrical power.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) <b>currentValue</b>
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YPower <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the electrical power

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**power**→**get\_errorMessage()****YPower****power**→**errorMessage()****power.get\_errorMessage()**

Returns the error message of the latest error with the electrical power sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the electrical power sensor object

**power**→**get\_errorType()****YPower****power**→**errorType()****power.get\_errorType()**

Returns the numerical error code of the latest error with the electrical power sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the electrical power sensor object

**power**→**get\_friendlyName()****YPower****power**→**friendlyName()****power.get\_friendlyName()**

Returns a global identifier of the electrical power sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the electrical power sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the electrical power sensor (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the electrical power sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.



**power**→**get\_functionDescriptor()****YPower****power**→**functionDescriptor()****power.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**power**→**get\_functionId()****YPower****power**→**functionId()****power.get\_functionId()**

Returns the hardware identifier of the electrical power sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the electrical power sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**power**→**get\_hardwareId()****YPower****power**→**hardwareId()****power.get\_hardwareId()**

Returns the unique hardware identifier of the electrical power sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the electrical power sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the electrical power sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**power**→**get\_highestValue()****YPower****power**→**highestValue()****power.get\_highestValue()**

Returns the maximal value observed for the electrical power.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YPower <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the electrical power

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**power→get\_logFrequency()****YPower****power→logFrequency()power.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YPower <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**power**→**get\_logicalName()****YPower****power**→**logicalName()****power.get\_logicalName()**

Returns the logical name of the electrical power sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YPower <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the electrical power sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**power**→**get\_lowestValue()****YPower****power**→**lowestValue()****power.get\_lowestValue()**

Returns the minimal value observed for the electrical power.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YPower <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the electrical power

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**power**→**get\_meter()****YPower****power**→**meter()****power.get\_meter()**

Returns the energy counter, maintained by the wattmeter by integrating the power consumption over time.

js	function <b>get_meter</b> ( )
nodejs	function <b>get_meter</b> ( )
php	function <b>get_meter</b> ( )
cpp	double <b>get_meter</b> ( )
m	-(double) meter
pas	function <b>get_meter</b> ( ): double
vb	function <b>get_meter</b> ( ) As Double
cs	double <b>get_meter</b> ( )
java	double <b>get_meter</b> ( )
py	def <b>get_meter</b> ( )
cmd	YPower <b>target</b> <b>get_meter</b>

Note that this counter is reset at each start of the device.

**Returns :**

a floating point number corresponding to the energy counter, maintained by the wattmeter by integrating the power consumption over time

On failure, throws an exception or returns Y\_METER\_INVALID.



**power**→**get\_meterTimer()****YPower****power**→**meterTimer()****power.get\_meterTimer()**

Returns the elapsed time since last energy counter reset, in seconds.

js	function <b>get_meterTimer</b> ( )
nodejs	function <b>get_meterTimer</b> ( )
php	function <b>get_meterTimer</b> ( )
cpp	int <b>get_meterTimer</b> ( )
m	-(int) meterTimer
pas	function <b>get_meterTimer</b> ( ): LongInt
vb	function <b>get_meterTimer</b> ( ) As Integer
cs	int <b>get_meterTimer</b> ( )
java	int <b>get_meterTimer</b> ( )
py	def <b>get_meterTimer</b> ( )
cmd	YPower <b>target</b> <b>get_meterTimer</b>

**Returns :**

an integer corresponding to the elapsed time since last energy counter reset, in seconds

On failure, throws an exception or returns Y\_METERTIMER\_INVALID.

**power**→**get\_module()****YPower****power**→**module()****power.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**power→get\_module\_async()****YPower****power→module\_async()**

---

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**power**→**get\_recordedData()****YPower****power**→**recordedData()****power.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YPower <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**power→get\_reportFrequency()****YPower****power→reportFrequency()****power.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YPower <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**power**→**get\_resolution()****YPower****power**→**resolution()****power.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YPower <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**power**→**get\_unit()****YPower****power**→**unit()****power.get\_unit()**

Returns the measuring unit for the electrical power.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YPower <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the electrical power

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**power**→**get\_userData()****power**→**userData()****power.getUserData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.



**power→isOnline()power.isOnline()****YPower**

Checks if the electrical power sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the electrical power sensor 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 electrical power sensor.

**Returns :**

`true` if the electrical power sensor can be reached, and `false` otherwise

**power→isOnline\_async()****YPower**

Checks if the electrical power sensor 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 electrical power sensor 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.

**power→load()power.load()****YPower**

Preloads the electrical power sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

## power→loadCalibrationPoints() power.loadCalibrationPoints()

YPower

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)
py def loadCalibrationPoints( rawValues, refValues)
cmd YPower target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power→load\_async()****YPower**

Preloads the electrical power sensor 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 traffic 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.

**power**→**nextPower()****power.nextPower()****YPower**

Continues the enumeration of electrical power sensors started using `yFirstPower()`.

js	function <b>nextPower</b> ( )
nodejs	function <b>nextPower</b> ( )
php	function <b>nextPower</b> ( )
cpp	YPower * <b>nextPower</b> ( )
m	-(YPower*) <b>nextPower</b>
pas	function <b>nextPower</b> ( ): TYPower
vb	function <b>nextPower</b> ( ) As YPower
cs	YPower <b>nextPower</b> ( )
java	YPower <b>nextPower</b> ( )
py	def <b>nextPower</b> ( )

**Returns :**

a pointer to a `YPower` object, corresponding to a electrical power sensor currently online, or a `null` pointer if there are no more electrical power sensors to enumerate.

## power→registerTimedReportCallback() power.registerTimedReportCallback()

YPower

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YPowerTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YPowerTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYPowerTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## power→registerValueCallback() power.registerValueCallback()

YPower

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YPowerValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YPowerValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYPowerValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.



**power→reset()****power.reset()****YPower**

Resets the energy counter.

js	function <b>reset</b> ( )
nodejs	function <b>reset</b> ( )
php	function <b>reset</b> ( )
cpp	int <b>reset</b> ( )
m	-(int) <b>reset</b>
pas	function <b>reset</b> ( ): LongInt
vb	function <b>reset</b> ( ) As Integer
cs	int <b>reset</b> ( )
java	int <b>reset</b> ( )
py	def <b>reset</b> ( )
cmd	YPower <b>target reset</b>

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power**→**set\_highestValue()****YPower****power**→**setHighestValue()****power.set\_highestValue()**

Changes the recorded maximal value observed pour the electrical power.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YPower <b>target set_highestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed pour the electrical power

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power→set\_logFrequency()****YPower****power→setLogFrequency()power.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YPower <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power**→**set\_logicalName()****YPower****power**→**setLogicalName()****power.set\_logicalName()**

Changes the logical name of the electrical power sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YPower <b>target set_logicalName newval</b>

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 electrical power sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**power**→**set\_lowestValue()****YPower****power**→**setLowestValue()****power.set\_lowestValue()**

Changes the recorded minimal value observed pour the electrical power.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YPower <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed pour the electrical power

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power**→**set\_reportFrequency()**  
**power**→**setReportFrequency()**  
**power.set\_reportFrequency()**

**YPower**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YPower <b>target set_reportFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power**→**set\_resolution()****YPower****power**→**setResolution()****power.set\_resolution()**

Changes the resolution of the measured values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YPower <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**power**→**set\_userData()****power**→**setUserData()****power.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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



**power→wait\_async()****YPower**

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.

## 3.29. Pressure function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_pressure.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPressure = yoctolib.YPressure;
php	require_once('yocto_pressure.php');
c++	#include "yocto_pressure.h"
m	#import "yocto_pressure.h"
pas	uses yocto_pressure;
vb	yocto_pressure.vb
cs	yocto_pressure.cs
java	import com.yoctopuce.YoctoAPI.YPressure;
py	from yocto_pressure import *

### Global functions

#### yFindPressure(func)

Retrieves a pressure sensor for a given identifier.

#### yFirstPressure()

Starts the enumeration of pressure sensors currently accessible.

### YPressure methods

#### pressure→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### pressure→describe()

Returns a short text that describes unambiguously the instance of the pressure sensor in the form `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### pressure→get\_advertisedValue()

Returns the current value of the pressure sensor (no more than 6 characters).

#### pressure→get\_currentRawValue()

Returns the unrounded and uncalibrated raw value returned by the sensor.

#### pressure→get\_currentValue()

Returns the current measure for the pressure.

#### pressure→get\_errorMessage()

Returns the error message of the latest error with the pressure sensor.

#### pressure→get\_errorType()

Returns the numerical error code of the latest error with the pressure sensor.

#### pressure→get\_friendlyName()

Returns a global identifier of the pressure sensor in the format `MODULE_NAME . FUNCTION_NAME`.

#### pressure→get\_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

#### pressure→get\_functionId()

Returns the hardware identifier of the pressure sensor, without reference to the module.

#### pressure→get\_hardwareId()

Returns the unique hardware identifier of the pressure sensor in the form `SERIAL . FUNCTIONID`.

**pressure→get\_highestValue()**

Returns the maximal value observed for the pressure.

**pressure→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**pressure→get\_logicalName()**

Returns the logical name of the pressure sensor.

**pressure→get\_lowestValue()**

Returns the minimal value observed for the pressure.

**pressure→get\_module()**

Gets the YModule object for the device on which the function is located.

**pressure→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**pressure→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**pressure→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**pressure→get\_resolution()**

Returns the resolution of the measured values.

**pressure→get\_unit()**

Returns the measuring unit for the pressure.

**pressure→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**pressure→isOnline()**

Checks if the pressure sensor is currently reachable, without raising any error.

**pressure→isOnline\_async(callback, context)**

Checks if the pressure sensor is currently reachable, without raising any error (asynchronous version).

**pressure→load(msValidity)**

Preloads the pressure sensor cache with a specified validity duration.

**pressure→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**pressure→load\_async(msValidity, callback, context)**

Preloads the pressure sensor cache with a specified validity duration (asynchronous version).

**pressure→nextPressure()**

Continues the enumeration of pressure sensors started using yFirstPressure().

**pressure→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**pressure→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**pressure→set\_highestValue(newval)**

Changes the recorded maximal value observed for the pressure.

**pressure→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**pressure→set\_logicalName(newval)**

Changes the logical name of the pressure sensor.

### 3. Reference

---

**pressure**→**set\_lowestValue**(newval)

Changes the recorded minimal value observed for the pressure.

**pressure**→**set\_reportFrequency**(newval)

Changes the timed value notification frequency for this function.

**pressure**→**set\_resolution**(newval)

Changes the resolution of the measured physical values.

**pressure**→**set\_userData**(data)

Stores a user context provided as argument in the userData attribute of the function.

**pressure**→**wait\_async**(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YPressure.FindPressure() yFindPressure()YPressure.FindPressure()

## YPressure

Retrieves a pressure sensor for a given identifier.

js	function <b>yFindPressure</b> ( <b>func</b> )
nodejs	function <b>FindPressure</b> ( <b>func</b> )
php	function <b>yFindPressure</b> ( <b>\$func</b> )
cpp	YPressure* <b>yFindPressure</b> ( const string& <b>func</b> )
m	YPressure* <b>yFindPressure</b> ( NSString* <b>func</b> )
pas	function <b>yFindPressure</b> ( <b>func</b> : string): TYPressure
vb	function <b>yFindPressure</b> ( ByVal <b>func</b> As String) As YPressure
cs	YPressure <b>FindPressure</b> ( string <b>func</b> )
java	YPressure <b>FindPressure</b> ( String <b>func</b> )
py	def <b>FindPressure</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the pressure sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YPressure.isOnline()` to test if the pressure sensor is indeed online at a given time. In case of ambiguity when looking for a pressure sensor 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 pressure sensor

### Returns :

a YPressure object allowing you to drive the pressure sensor.

**YPressure.FirstPressure()****YPressure****yFirstPressure()YPressure.FirstPressure()**

Starts the enumeration of pressure sensors currently accessible.

js	function <b>yFirstPressure</b> ( )
nodejs	function <b>FirstPressure</b> ( )
php	function <b>yFirstPressure</b> ( )
cpp	YPressure* <b>yFirstPressure</b> ( )
m	YPressure* <b>yFirstPressure</b> ( )
pas	function <b>yFirstPressure</b> ( ): TYPresure
vb	function <b>yFirstPressure</b> ( ) As YPressure
cs	YPressure <b>FirstPressure</b> ( )
java	YPressure <b>FirstPressure</b> ( )
py	def <b>FirstPressure</b> ( )

Use the method `YPressure.nextPressure( )` to iterate on next pressure sensors.

**Returns :**

a pointer to a YPressure object, corresponding to the first pressure sensor currently online, or a `null` pointer if there are none.

## pressure→calibrateFromPoints() pressure.calibrateFromPoints()

YPressure

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

js	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
nodejs	function <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
php	function <b>calibrateFromPoints</b> ( <b>\$rawValues</b> , <b>\$refValues</b> )
cpp	int <b>calibrateFromPoints</b> ( vector<double> <b>rawValues</b> , vector<double> <b>refValues</b> )
m	-(int) <b>calibrateFromPoints</b> : (NSMutableArray*) <b>rawValues</b> : (NSMutableArray*) <b>refValues</b>
pas	function <b>calibrateFromPoints</b> ( <b>rawValues</b> : TDoubleArray, <b>refValues</b> : TDoubleArray): LongInt
vb	procedure <b>calibrateFromPoints</b> ( )
cs	int <b>calibrateFromPoints</b> ( List<double> <b>rawValues</b> , List<double> <b>refValues</b> )
java	int <b>calibrateFromPoints</b> ( ArrayList<Double> <b>rawValues</b> , ArrayList<Double> <b>refValues</b> )
py	def <b>calibrateFromPoints</b> ( <b>rawValues</b> , <b>refValues</b> )
cmd	YPressure <b>target</b> <b>calibrateFromPoints</b> <b>rawValues</b> <b>refValues</b>

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pressure→describe()****pressure.describe()****YPressure**

Returns a short text that describes unambiguously the instance of the pressure sensor in the form  
 TYPE ( NAME ) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 pressure sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)



**pressure**→**get\_advertisedValue()****YPressure****pressure**→**advertisedValue()****pressure.get\_advertisedValue()**

Returns the current value of the pressure sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YPressure <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the pressure sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**pressure**→**get\_currentRawValue()****YPressure****pressure**→**currentRawValue()****pressure.get\_currentRawValue()**

Returns the unrounded and uncalibrated raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YPressure <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the unrounded and uncalibrated raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**pressure**→**get\_currentValue()****YPressure****pressure**→**currentValue()****pressure.get\_currentValue()**

Returns the current measure for the pressure.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YPressure <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the pressure

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**pressure**→**get\_errorMessage()****YPressure****pressure**→**errorMessage()****pressure.get\_errorMessage()**

Returns the error message of the latest error with the pressure sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the pressure sensor object

---

**pressure**→**get\_errorType()****YPressure****pressure**→**errorType()****pressure.get\_errorType()**

---

Returns the numerical error code of the latest error with the pressure sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the pressure sensor object

**pressure**→**get\_friendlyName()****YPressure****pressure**→**friendlyName()****pressure.get\_friendlyName()**

Returns a global identifier of the pressure sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the pressure sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the pressure sensor (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the pressure sensor using logical names (ex: `MyCustomName.relay1`)

On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**pressure**→**get\_functionDescriptor()**  
**pressure**→**functionDescriptor()**  
**pressure.get\_functionDescriptor()**

**YPressure**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**pressure**→**get\_functionId()****YPressure****pressure**→**functionId()****pressure.get\_functionId()**

Returns the hardware identifier of the pressure sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the pressure sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.



**pressure**→**get\_hardwareId()****YPressure****pressure**→**hardwareId()****pressure.get\_hardwareId()**

Returns the unique hardware identifier of the pressure sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the pressure sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the pressure sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**pressure**→**get\_highestValue()****YPressure****pressure**→**highestValue()****pressure.get\_highestValue()**

Returns the maximal value observed for the pressure.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YPressure <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the pressure

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**pressure→get\_logFrequency()****YPressure****pressure→logFrequency()****pressure.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YPressure <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**pressure**→**get\_logicalName()****YPressure****pressure**→**logicalName()****pressure.get\_logicalName()**

Returns the logical name of the pressure sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YPressure <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the pressure sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**pressure**→**get\_lowestValue()****YPressure****pressure**→**lowestValue()****pressure.get\_lowestValue()**

Returns the minimal value observed for the pressure.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YPressure <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the pressure

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**pressure**→**get\_module()****YPressure****pressure**→**module()****pressure.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**pressure→get\_module\_async()****YPressure****pressure→module\_async()**

---

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**pressure→get\_recordedData()****YPressure****pressure→recordedData()****pressure.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
c++	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YPressure <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.



**pressure→get\_reportFrequency()****YPressure****pressure→reportFrequency()****pressure.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YPressure <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**pressure**→**get\_resolution()****YPressure****pressure**→**resolution()****pressure.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YPressure <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**pressure**→**get\_unit()****YPressure****pressure**→**unit()****pressure.get\_unit()**

Returns the measuring unit for the pressure.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YPressure <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the pressure

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**pressure**→**get\_userData()****YPressure****pressure**→**userData()****pressure.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**pressure**→**isOnline()****pressure.isOnline()****YPressure**

Checks if the pressure sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the pressure sensor 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 pressure sensor.

**Returns :**

`true` if the pressure sensor can be reached, and `false` otherwise

**pressure→isOnline\_async()****YPressure**

Checks if the pressure sensor 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 pressure sensor 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.

**pressure→load()pressure.load()****YPressure**

Preloads the pressure sensor cache with a specified validity duration.

js	function load( <b>msValidity</b> )
nodejs	function load( <b>msValidity</b> )
php	function load( <b>\$msValidity</b> )
cpp	YRETCODE load( int <b>msValidity</b> )
m	-(YRETCODE) load : (int) <b>msValidity</b>
pas	function load( <b>msValidity</b> : integer): YRETCODE
vb	function load( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE load( int <b>msValidity</b> )
java	int load( long <b>msValidity</b> )
py	def load( <b>msValidity</b> )

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 traffic 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.

## pressure→loadCalibrationPoints() pressure.loadCalibrationPoints()

YPressure

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)
py def loadCalibrationPoints( rawValues, refValues)
cmd YPressure target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**pressure→load\_async()****YPressure**

Preloads the pressure sensor 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 traffic 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.

**pressure**→**nextPressure()****pressure.nextPressure()****YPressure**

Continues the enumeration of pressure sensors started using `yFirstPressure()`.

js	function <b>nextPressure</b> ( )
nodejs	function <b>nextPressure</b> ( )
php	function <b>nextPressure</b> ( )
cpp	YPressure * <b>nextPressure</b> ( )
m	-(YPressure*) <b>nextPressure</b>
pas	function <b>nextPressure</b> ( ): TYPressure
vb	function <b>nextPressure</b> ( ) As YPressure
cs	YPressure <b>nextPressure</b> ( )
java	YPressure <b>nextPressure</b> ( )
py	def <b>nextPressure</b> ( )

**Returns :**

a pointer to a YPressure object, corresponding to a pressure sensor currently online, or a `null` pointer if there are no more pressure sensors to enumerate.

## pressure→registerTimedReportCallback() pressure.registerTimedReportCallback()

YPressure

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YPressureTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YPressureTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYPressureTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## pressure→registerValueCallback() pressure.registerValueCallback()

YPressure

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YPressureValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YPressureValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYPressureValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**pressure**→**set\_highestValue()****YPressure****pressure**→**setHighestValue()****pressure.set\_highestValue()**

Changes the recorded maximal value observed for the pressure.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YPressure <b>target</b> <b>set_highestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed for the pressure

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pressure→set\_logFrequency()****YPressure****pressure→setLogFrequency()****pressure.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

<code>js</code>	<code>function set_logFrequency( newval)</code>
<code>nodejs</code>	<code>function set_logFrequency( newval)</code>
<code>php</code>	<code>function set_logFrequency( \$newval)</code>
<code>cpp</code>	<code>int set_logFrequency( const string&amp; newval)</code>
<code>m</code>	<code>-(int) setLogFrequency : (NSString*) newval</code>
<code>pas</code>	<code>function set_logFrequency( newval: string): integer</code>
<code>vb</code>	<code>function set_logFrequency( ByVal newval As String) As Integer</code>
<code>cs</code>	<code>int set_logFrequency( string newval)</code>
<code>java</code>	<code>int set_logFrequency( String newval)</code>
<code>py</code>	<code>def set_logFrequency( newval)</code>
<code>cmd</code>	<code>YPressure target set_logFrequency newval</code>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pressure→set\_logicalName()****YPressure****pressure→setLogicalName()****pressure.set\_logicalName()**

Changes the logical name of the pressure sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YPressure <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 pressure sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**pressure**→**set\_lowestValue()****YPressure****pressure**→**setLowestValue()****pressure.set\_lowestValue()**

Changes the recorded minimal value observed for the pressure.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YPressure <b>target</b> <b>set_lowestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed for the pressure

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**pressure→set\_reportFrequency()**  
**pressure→setReportFrequency()**  
**pressure.set\_reportFrequency()**

**YPressure**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YPressure <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pressure**→**set\_resolution()****YPressure****pressure**→**setResolution()****pressure.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YPressure <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pressure**→**set\_userData()****YPressure****pressure**→**setUserData()****pressure.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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

**pressure**→**wait\_async()****YPressure**

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.

## 3.30. Pwm function interface

The Yoctopuce application programming interface allows you to configure, start, and stop the PWM.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_pwmoutput.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPwmOutput = yoctolib.YPwmOutput;
php	require_once('yocto_pwmoutput.php');
c++	#include "yocto_pwmoutput.h"
m	#import "yocto_pwmoutput.h"
pas	uses yocto_pwmoutput;
vb	yocto_pwmoutput.vb
cs	yocto_pwmoutput.cs
java	import com.yoctopuce.YoctoAPI.YPwmOutput;
py	from yocto_pwmoutput import *

### Global functions

#### yFindPwmOutput(func)

Retrieves a PWM for a given identifier.

#### yFirstPwmOutput()

Starts the enumeration of PWMs currently accessible.

### YPwmOutput methods

#### pwmoutput→describe()

Returns a short text that describes unambiguously the instance of the PWM in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### pwmoutput→dutyCycleMove(target, ms\_duration)

Performs a smooth change of the pulse duration toward a given value.

#### pwmoutput→get\_advertisedValue()

Returns the current value of the PWM (no more than 6 characters).

#### pwmoutput→get\_dutyCycle()

Returns the PWMs duty cycle as a floating point number between 0 and 1.

#### pwmoutput→get\_dutyCycleAtPowerOn()

Returns the PWMs duty cycle at device power up as a floating point number between 0.0 and 100.

#### pwmoutput→get\_enabled()

Returns the state of the PWMs.

#### pwmoutput→get\_enabledAtPowerOn()

Returns the state of the PWMs at device power up.

#### pwmoutput→get\_errorMessage()

Returns the error message of the latest error with the PWM.

#### pwmoutput→get\_errorType()

Returns the numerical error code of the latest error with the PWM.

#### pwmoutput→get\_frequency()

Returns the PWM frequency in Hz.

#### pwmoutput→get\_friendlyName()

Returns a global identifier of the PWM in the format MODULE\_NAME . FUNCTION\_NAME.

#### pwmoutput→get\_functionDescriptor()

	Returns a unique identifier of type <code>YFUN_DESCR</code> corresponding to the function.
<b><code>pwmoutput→get_functionId()</code></b>	Returns the hardware identifier of the PWM, without reference to the module.
<b><code>pwmoutput→get_hardwareId()</code></b>	Returns the unique hardware identifier of the PWM in the form <code>SERIAL . FUNCTIONID</code> .
<b><code>pwmoutput→get_logicalName()</code></b>	Returns the logical name of the PWM.
<b><code>pwmoutput→get_module()</code></b>	Gets the <code>YModule</code> object for the device on which the function is located.
<b><code>pwmoutput→get_module_async(callback, context)</code></b>	Gets the <code>YModule</code> object for the device on which the function is located (asynchronous version).
<b><code>pwmoutput→get_period()</code></b>	Returns the PWM period in nanoseconds.
<b><code>pwmoutput→get_pulseDuration()</code></b>	Returns the PWM pulse length in milliseconds.
<b><code>pwmoutput→get_userData()</code></b>	Returns the value of the <code>userData</code> attribute, as previously stored using method <code>set_userData</code> .
<b><code>pwmoutput→isOnline()</code></b>	Checks if the PWM is currently reachable, without raising any error.
<b><code>pwmoutput→isOnline_async(callback, context)</code></b>	Checks if the PWM is currently reachable, without raising any error (asynchronous version).
<b><code>pwmoutput→load(msValidity)</code></b>	Preloads the PWM cache with a specified validity duration.
<b><code>pwmoutput→load_async(msValidity, callback, context)</code></b>	Preloads the PWM cache with a specified validity duration (asynchronous version).
<b><code>pwmoutput→nextPwmOutput()</code></b>	Continues the enumeration of PWMs started using <code>yFirstPwmOutput()</code> .
<b><code>pwmoutput→pulseDurationMove(ms_target, ms_duration)</code></b>	Performs a smooth change of the pulse duration toward a given value.
<b><code>pwmoutput→registerValueCallback(callback)</code></b>	Registers the callback function that is invoked on every change of advertised value.
<b><code>pwmoutput→set_dutyCycle(newval)</code></b>	Configures the PWMs duty cycle.
<b><code>pwmoutput→set_dutyCycleAtPowerOn(newval)</code></b>	Configures the PWMs duty cycle at device power up.
<b><code>pwmoutput→set_enabled(newval)</code></b>	Stops or starts the PWM.
<b><code>pwmoutput→set_enabledAtPowerOn(newval)</code></b>	Configures the state of PWM at device power up.
<b><code>pwmoutput→set_frequency(newval)</code></b>	Configures the PWM frequency.
<b><code>pwmoutput→set_logicalName(newval)</code></b>	Changes the logical name of the PWM.
<b><code>pwmoutput→set_period(newval)</code></b>	Configures the PWM period.

**pwmoutput→set\_pulseDuration(newval)**

Configures the PWM pluses length.

**pwmoutput→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**pwmoutput→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YPwmOutput.FindPwmOutput()****YPwmOutput****yFindPwmOutput()** **YPwmOutput.FindPwmOutput()**

Retrieves a PWM for a given identifier.

js	function <b>yFindPwmOutput</b> ( <b>func</b> )
nodejs	function <b>FindPwmOutput</b> ( <b>func</b> )
php	function <b>yFindPwmOutput</b> ( <b>\$func</b> )
cpp	YPwmOutput* <b>yFindPwmOutput</b> ( const string& <b>func</b> )
m	YPwmOutput* <b>yFindPwmOutput</b> ( NSString* <b>func</b> )
pas	function <b>yFindPwmOutput</b> ( <b>func</b> : string): TYPwmOutput
vb	function <b>yFindPwmOutput</b> ( ByVal <b>func</b> As String) As YPwmOutput
cs	YPwmOutput <b>FindPwmOutput</b> ( string <b>func</b> )
java	YPwmOutput <b>FindPwmOutput</b> ( String <b>func</b> )
py	def <b>FindPwmOutput</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the PWM is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YPwmOutput.isOnline()` to test if the PWM is indeed online at a given time. In case of ambiguity when looking for a PWM 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 PWM

**Returns :**

a `YPwmOutput` object allowing you to drive the PWM.



**YPwmOutput.FirstPwmOutput()****YPwmOutput****yFirstPwmOutput()YPwmOutput.FirstPwmOutput()**

Starts the enumeration of PWMs currently accessible.

<code>js</code>	<code>function yFirstPwmOutput( )</code>
<code>nodejs</code>	<code>function FirstPwmOutput( )</code>
<code>php</code>	<code>function yFirstPwmOutput( )</code>
<code>cpp</code>	<code>YPwmOutput* yFirstPwmOutput( )</code>
<code>m</code>	<code>YPwmOutput* yFirstPwmOutput( )</code>
<code>pas</code>	<code>function yFirstPwmOutput( ): TYPwmOutput</code>
<code>vb</code>	<code>function yFirstPwmOutput( ) As YPwmOutput</code>
<code>cs</code>	<code>YPwmOutput FirstPwmOutput( )</code>
<code>java</code>	<code>YPwmOutput FirstPwmOutput( )</code>
<code>py</code>	<code>def FirstPwmOutput( )</code>

Use the method `YPwmOutput.nextPwmOutput( )` to iterate on next PWMs.

**Returns :**

a pointer to a `YPwmOutput` object, corresponding to the first PWM currently online, or a `null` pointer if there are none.

**pwmoutput→describe()**pwmoutput.describe()**YPwmOutput**

Returns a short text that describes unambiguously the instance of the PWM in the form  
 TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 PWM (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

## pwmoutput→dutyCycleMove() pwmoutput.dutyCycleMove()

## YPwmOutput

Performs a smooth change of the pulse duration toward a given value.

js	function <b>dutyCycleMove</b> ( <b>target</b> , <b>ms_duration</b> )
nodejs	function <b>dutyCycleMove</b> ( <b>target</b> , <b>ms_duration</b> )
php	function <b>dutyCycleMove</b> ( <b>\$target</b> , <b>\$ms_duration</b> )
cpp	int <b>dutyCycleMove</b> ( double <b>target</b> , int <b>ms_duration</b> )
m	-(int) <b>dutyCycleMove</b> : (double) <b>target</b> : (int) <b>ms_duration</b>
pas	function <b>dutyCycleMove</b> ( <b>target</b> : double, <b>ms_duration</b> : LongInt): LongInt
vb	function <b>dutyCycleMove</b> ( ) As Integer
cs	int <b>dutyCycleMove</b> ( double <b>target</b> , int <b>ms_duration</b> )
java	int <b>dutyCycleMove</b> ( double <b>target</b> , int <b>ms_duration</b> )
py	def <b>dutyCycleMove</b> ( <b>target</b> , <b>ms_duration</b> )
cmd	YPwmOutput <b>target</b> <b>dutyCycleMove</b> <b>target</b> <b>ms_duration</b>

### Parameters :

- target** new duty cycle at the end of the transition (floating-point number, between 0 and 1)
- ms\_duration** total duration of the transition, in milliseconds

### Returns :

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**pwmoutput→get\_advertisedValue()****YPwmOutput****pwmoutput→advertisedValue()****pwmoutput.get\_advertisedValue()**

Returns the current value of the PWM (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the PWM (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**pwmoutput→get\_dutyCycle()****YPwmOutput****pwmoutput→dutyCycle()****pwmoutput.get\_dutyCycle()**

Returns the PWMs dutty cyle as a floating point number between 0 an 1.

js	function <b>get_dutyCycle</b> ( )
nodejs	function <b>get_dutyCycle</b> ( )
php	function <b>get_dutyCycle</b> ( )
cpp	double <b>get_dutyCycle</b> ( )
m	-(double) dutyCycle
pas	function <b>get_dutyCycle</b> ( ): double
vb	function <b>get_dutyCycle</b> ( ) As Double
cs	double <b>get_dutyCycle</b> ( )
java	double <b>get_dutyCycle</b> ( )
py	def <b>get_dutyCycle</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_dutyCycle</b>

**Returns :**

a floating point number corresponding to the PWMs dutty cyle as a floating point number between 0 an 1

On failure, throws an exception or returns Y\_DUTYCYCLE\_INVALID.

**pwmoutput→get\_dutyCycleAtPowerOn()**  
**pwmoutput→dutyCycleAtPowerOn()**  
**pwmoutput.get\_dutyCycleAtPowerOn()**

**YPwmOutput**

Returns the PWMs duty cycle at device power up as a floating point number between 0.0 and 100.

js	function <b>get_dutyCycleAtPowerOn</b> ( )
nodejs	function <b>get_dutyCycleAtPowerOn</b> ( )
php	function <b>get_dutyCycleAtPowerOn</b> ( )
cpp	double <b>get_dutyCycleAtPowerOn</b> ( )
m	-(double) dutyCycleAtPowerOn
pas	function <b>get_dutyCycleAtPowerOn</b> ( ): double
vb	function <b>get_dutyCycleAtPowerOn</b> ( ) As Double
cs	double <b>get_dutyCycleAtPowerOn</b> ( )
java	double <b>get_dutyCycleAtPowerOn</b> ( )
py	def <b>get_dutyCycleAtPowerOn</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_dutyCycleAtPowerOn</b>

0%

**Returns :**

a floating point number corresponding to the PWMs duty cycle at device power up as a floating point number between 0.0 and 100

On failure, throws an exception or returns Y\_DUTYCYCLEATPOWERON\_INVALID.

**pwmoutput→get\_enabled()****YPwmOutput****pwmoutput→enabled()pwmoutput.get\_enabled()**

Returns the state of the PWMs.

js	function <b>get_enabled</b> ( )
nodejs	function <b>get_enabled</b> ( )
php	function <b>get_enabled</b> ( )
cpp	Y_ENABLED_enum <b>get_enabled</b> ( )
m	-(Y_ENABLED_enum) enabled
pas	function <b>get_enabled</b> ( ): Integer
vb	function <b>get_enabled</b> ( ) As Integer
cs	int <b>get_enabled</b> ( )
java	int <b>get_enabled</b> ( )
py	def <b>get_enabled</b> ( )
cmd	YPwmOutput <b>target get_enabled</b>

**Returns :**

either Y\_ENABLED\_FALSE or Y\_ENABLED\_TRUE, according to the state of the PWMs

On failure, throws an exception or returns Y\_ENABLED\_INVALID.

**pwmoutput→get\_enabledAtPowerOn()****YPwmOutput****pwmoutput→enabledAtPowerOn()****pwmoutput.get\_enabledAtPowerOn()**

Returns the state of the PWMs at device power up.

js	function <b>get_enabledAtPowerOn</b> ( )
nodejs	function <b>get_enabledAtPowerOn</b> ( )
php	function <b>get_enabledAtPowerOn</b> ( )
cpp	Y_ENABLEDATPOWERON_enum <b>get_enabledAtPowerOn</b> ( )
m	-(Y_ENABLEDATPOWERON_enum) enabledAtPowerOn
pas	function <b>get_enabledAtPowerOn</b> ( ): Integer
vb	function <b>get_enabledAtPowerOn</b> ( ) As Integer
cs	int <b>get_enabledAtPowerOn</b> ( )
java	int <b>get_enabledAtPowerOn</b> ( )
py	def <b>get_enabledAtPowerOn</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_enabledAtPowerOn</b>

**Returns :**

either Y\_ENABLEDATPOWERON\_FALSE or Y\_ENABLEDATPOWERON\_TRUE, according to the state of the PWMs at device power up

On failure, throws an exception or returns Y\_ENABLEDATPOWERON\_INVALID.



---

**pwmoutput→get\_errorMessage()****YPwmOutput****pwmoutput→errorMessage()****pwmoutput.get\_errorMessage()**

---

Returns the error message of the latest error with the PWM.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the PWM object

**pwmoutput→get\_errorType()****YPwmOutput****pwmoutput→errorType()****pwmoutput.get\_errorType()**

Returns the numerical error code of the latest error with the PWM.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the PWM object

**pwmoutput→get\_frequency()****YPwmOutput****pwmoutput→frequency()****pwmoutput.get\_frequency()**

Returns the PWM frequency in Hz.

js	function <b>get_frequency</b> ( )
nodejs	function <b>get_frequency</b> ( )
php	function <b>get_frequency</b> ( )
cpp	int <b>get_frequency</b> ( )
m	-(int) frequency
pas	function <b>get_frequency</b> ( ): LongInt
vb	function <b>get_frequency</b> ( ) As Integer
cs	int <b>get_frequency</b> ( )
java	int <b>get_frequency</b> ( )
py	def <b>get_frequency</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_frequency</b>

**Returns :**

an integer corresponding to the PWM frequency in Hz

On failure, throws an exception or returns Y\_FREQUENCY\_INVALID.

**pwmoutput→get\_friendlyName()****YPwmOutput****pwmoutput→friendlyName()****pwmoutput.get\_friendlyName()**

Returns a global identifier of the PWM in the format `MODULE_NAME . FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the PWM if they are defined, otherwise the serial number of the module and the hardware identifier of the PWM (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the PWM using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**pwmoutput→get\_functionDescriptor()**  
**pwmoutput→functionDescriptor()**  
**pwmoutput.get\_functionDescriptor()**

**YPwmOutput**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**pwmoutput→get\_functionId()****YPwmOutput****pwmoutput→functionId()pwmoutput.get\_functionId()**

Returns the hardware identifier of the PWM, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) functionId
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the PWM (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**pwmoutput→get\_hardwareId()****YPwmOutput****pwmoutput→hardwareId()****pwmoutput.get\_hardwareId()**

Returns the unique hardware identifier of the PWM in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the PWM. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the PWM (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**pwmoutput→get\_logicalName()****YPwmOutput****pwmoutput→logicalName()****pwmoutput.get\_logicalName()**

Returns the logical name of the PWM.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the PWM. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**pwmoutput→get\_module()****YPwmOutput****pwmoutput→module()pwmoutput.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**pwmoutput→get\_module\_async()**  
**pwmoutput→module\_async()****YPwmOutput**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

---

**pwmoutput→get\_period()****YPwmOutput****pwmoutput→period()pwmoutput.get\_period()**

---

Returns the PWM period in nonaseconde.

js	function <b>get_period</b> ( )
nodejs	function <b>get_period</b> ( )
php	function <b>get_period</b> ( )
cpp	double <b>get_period</b> ( )
m	-(double) period
pas	function <b>get_period</b> ( ): double
vb	function <b>get_period</b> ( ) As Double
cs	double <b>get_period</b> ( )
java	double <b>get_period</b> ( )
py	def <b>get_period</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_period</b>

**Returns :**

a floating point number corresponding to the PWM period in nonaseconde

On failure, throws an exception or returns Y\_PERIOD\_INVALID.

**pwmoutput→get\_pulseDuration()**  
**pwmoutput→pulseDuration()**  
**pwmoutput.get\_pulseDuration()**

**YPwmOutput**

Returns the PWM pulse length in milliseconds.

js	function <b>get_pulseDuration</b> ( )
nodejs	function <b>get_pulseDuration</b> ( )
php	function <b>get_pulseDuration</b> ( )
cpp	double <b>get_pulseDuration</b> ( )
m	-(double) pulseDuration
pas	function <b>get_pulseDuration</b> ( ): double
vb	function <b>get_pulseDuration</b> ( ) As Double
cs	double <b>get_pulseDuration</b> ( )
java	double <b>get_pulseDuration</b> ( )
py	def <b>get_pulseDuration</b> ( )
cmd	YPwmOutput <b>target</b> <b>get_pulseDuration</b>

**Returns :**

a floating point number corresponding to the PWM pulse length in milliseconds

On failure, throws an exception or returns Y\_PULSEDURATION\_INVALID.

**pwmoutput→get\_userdata()****YPwmOutput****pwmoutput→userdata()pwmoutput.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**pwmoutput→isOnline()****pwmoutput.isOnline()****YPwmOutput**

Checks if the PWM is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the PWM 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 PWM.

**Returns :**

`true` if the PWM can be reached, and `false` otherwise

---

**pwmoutput→isOnline\_async()****YPwmOutput**

---

Checks if the PWM 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 PWM 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.

**pwmoutput→load()pwmoutput.load()****YPwmOutput**

Preloads the PWM cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.



**pwmoutput→load\_async()****YPwmOutput**

Preloads the PWM 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 traffic 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.

**pwmoutput→nextPwmOutput()**  
**pwmoutput.nextPwmOutput()****YPwmOutput**

Continues the enumeration of PWMs started using `yFirstPwmOutput()`.

js	function <b>nextPwmOutput</b> ( )
nodejs	function <b>nextPwmOutput</b> ( )
php	function <b>nextPwmOutput</b> ( )
c++	YPwmOutput * <b>nextPwmOutput</b> ( )
m	-(YPwmOutput*) <b>nextPwmOutput</b>
pas	function <b>nextPwmOutput</b> ( ): TYPwmOutput
vb	function <b>nextPwmOutput</b> ( ) As YPwmOutput
cs	YPwmOutput <b>nextPwmOutput</b> ( )
java	YPwmOutput <b>nextPwmOutput</b> ( )
py	def <b>nextPwmOutput</b> ( )

**Returns :**

a pointer to a `YPwmOutput` object, corresponding to a PWM currently online, or a `null` pointer if there are no more PWMs to enumerate.

## pwmoutput→pulseDurationMove() pwmoutput.pulseDurationMove()

YPwmOutput

Performs a smooth change of the pulse duration toward a given value.

js	function <b>pulseDurationMove</b> ( <b>ms_target</b> , <b>ms_duration</b> )
nodejs	function <b>pulseDurationMove</b> ( <b>ms_target</b> , <b>ms_duration</b> )
php	function <b>pulseDurationMove</b> ( <b>\$ms_target</b> , <b>\$ms_duration</b> )
cpp	int <b>pulseDurationMove</b> ( double <b>ms_target</b> , int <b>ms_duration</b> )
m	-(int) <b>pulseDurationMove</b> : (double) <b>ms_target</b> : (int) <b>ms_duration</b>
pas	function <b>pulseDurationMove</b> ( <b>ms_target</b> : double, <b>ms_duration</b> : LongInt): LongInt
vb	function <b>pulseDurationMove</b> ( ) As Integer
cs	int <b>pulseDurationMove</b> ( double <b>ms_target</b> , int <b>ms_duration</b> )
java	int <b>pulseDurationMove</b> ( double <b>ms_target</b> , int <b>ms_duration</b> )
py	def <b>pulseDurationMove</b> ( <b>ms_target</b> , <b>ms_duration</b> )
cmd	YPwmOutput <b>target pulseDurationMove ms_target ms_duration</b>

### Parameters :

- ms\_target** new pulse duration at the end of the transition (floating-point number, representing the pulse duration in milliseconds)
- ms\_duration** total duration of the transition, in milliseconds

### Returns :

YAPI\_SUCCESS when the call succeeds. On failure, throws an exception or returns a negative error code.

**pwmoutput→registerValueCallback()****YPwmOutput****pwmoutput.registerValueCallback()**

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YPwmOutputValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YPwmOutputValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYPwmOutputValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**pwmoutput→set\_dutyCycle()**  
**pwmoutput→setDutyCycle()**  
**pwmoutput.set\_dutyCycle()**

**YPwmOutput**

Configures the PWMs duty cyle.

js	function <b>set_dutyCycle</b> ( <b>newval</b> )
nodejs	function <b>set_dutyCycle</b> ( <b>newval</b> )
php	function <b>set_dutyCycle</b> ( <b>\$newval</b> )
cpp	int <b>set_dutyCycle</b> ( double <b>newval</b> )
m	-(int) setDutyCycle : (double) <b>newval</b>
pas	function <b>set_dutyCycle</b> ( <b>newval</b> : double): integer
vb	function <b>set_dutyCycle</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_dutyCycle</b> ( double <b>newval</b> )
java	int <b>set_dutyCycle</b> ( double <b>newval</b> )
py	def <b>set_dutyCycle</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target</b> <b>set_dutyCycle</b> <b>newval</b>

#### Parameters :

**newval** a floating point number

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pwmoutput→set\_dutyCycleAtPowerOn()**  
**pwmoutput→setDutyCycleAtPowerOn()**  
**pwmoutput.set\_dutyCycleAtPowerOn()**

**YPwmOutput**

Configures the PWMs duty cycle at device power up.

js	function <b>set_dutyCycleAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_dutyCycleAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_dutyCycleAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_dutyCycleAtPowerOn</b> ( double <b>newval</b> )
m	-(int) setDutyCycleAtPowerOn : (double) <b>newval</b>
pas	function <b>set_dutyCycleAtPowerOn</b> ( <b>newval</b> : double): integer
vb	function <b>set_dutyCycleAtPowerOn</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_dutyCycleAtPowerOn</b> ( double <b>newval</b> )
java	int <b>set_dutyCycleAtPowerOn</b> ( double <b>newval</b> )
py	def <b>set_dutyCycleAtPowerOn</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target</b> <b>set_dutyCycleAtPowerOn</b> <b>newval</b>

Remember to call the matching module `saveToFlash()` method, otherwise this call will have no effect.

#### Parameters :

**newval** a floating point number

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pwmoutput→set\_enabled()****YPwmOutput****pwmoutput→setEnabled()pwmoutput.set\_enabled()**

Stops or starts the PWM.

js	function <b>set_enabled</b> ( <b>newval</b> )
nodejs	function <b>set_enabled</b> ( <b>newval</b> )
php	function <b>set_enabled</b> ( <b>\$newval</b> )
cpp	int <b>set_enabled</b> ( Y_ENABLED_enum <b>newval</b> )
m	-(int) <b>setEnabled</b> : (Y_ENABLED_enum) <b>newval</b>
pas	function <b>set_enabled</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_enabled</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_enabled</b> ( int <b>newval</b> )
java	int <b>set_enabled</b> ( int <b>newval</b> )
py	def <b>set_enabled</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target set_enabled newval</b>

**Parameters :****newval** either Y\_ENABLED\_FALSE or Y\_ENABLED\_TRUE**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pwmoutput→set\_enabledAtPowerOn()**  
**pwmoutput→setEnabledAtPowerOn()**  
**pwmoutput.set\_enabledAtPowerOn()**

**YPwmOutput**

Configures the state of PWM at device power up.

js	function <b>set_enabledAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_enabledAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_enabledAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_enabledAtPowerOn</b> ( Y_ENABLEDATPOWERON_enum <b>newval</b> )
m	-(int) <b>setEnabledAtPowerOn</b> : (Y_ENABLEDATPOWERON_enum) <b>newval</b>
pas	function <b>set_enabledAtPowerOn</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_enabledAtPowerOn</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_enabledAtPowerOn</b> ( int <b>newval</b> )
java	int <b>set_enabledAtPowerOn</b> ( int <b>newval</b> )
py	def <b>set_enabledAtPowerOn</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target</b> <b>set_enabledAtPowerOn</b> <b>newval</b>

Remember to call the matching module `saveToFlash()` method, otherwise this call will have no effect.

#### Parameters :

**newval** either Y\_ENABLEDATPOWERON\_FALSE or Y\_ENABLEDATPOWERON\_TRUE

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**pwmoutput→set\_frequency()**  
**pwmoutput→setFrequency()**  
**pwmoutput.set\_frequency()**

**YPwmOutput**

Configures the PWM frequency.

js	function <b>set_frequency</b> ( <b>newval</b> )
nodejs	function <b>set_frequency</b> ( <b>newval</b> )
php	function <b>set_frequency</b> ( <b>\$newval</b> )
cpp	int <b>set_frequency</b> ( int <b>newval</b> )
m	-(int) setFrequency : (int) <b>newval</b>
pas	function <b>set_frequency</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_frequency</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_frequency</b> ( int <b>newval</b> )
java	int <b>set_frequency</b> ( int <b>newval</b> )
py	def <b>set_frequency</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target</b> <b>set_frequency</b> <b>newval</b>

The duty cycle is kept unchanged thanks to an automatic pulse width change.

#### Parameters :

**newval** an integer

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pwmoutput→set\_logicalName()**

**YPwmOutput**

**pwmoutput→setLogicalName()**

**pwmoutput.set\_logicalName()**

Changes the logical name of the PWM.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 PWM.

#### Returns :

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**pwmoutput→set\_period()****YPwmOutput****pwmoutput→setPeriod()****pwmoutput.set\_period()**

Configures the PWM period.

js	function <b>set_period</b> ( <b>newval</b> )
nodejs	function <b>set_period</b> ( <b>newval</b> )
php	function <b>set_period</b> ( <b>\$newval</b> )
cpp	int <b>set_period</b> ( double <b>newval</b> )
m	-(int) setPeriod : (double) <b>newval</b>
pas	function <b>set_period</b> ( <b>newval</b> : double): integer
vb	function <b>set_period</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_period</b> ( double <b>newval</b> )
java	int <b>set_period</b> ( double <b>newval</b> )
py	def <b>set_period</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target set_period newval</b>

**Parameters :****newval** a floating point number**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

pwmoutput→set\_pulseDuration()

YPwmOutput

pwmoutput→setPulseDuration()

pwmoutput.set\_pulseDuration()

Configures the PWM pluses length.

js	function <b>set_pulseDuration</b> ( <b>newval</b> )
nodejs	function <b>set_pulseDuration</b> ( <b>newval</b> )
php	function <b>set_pulseDuration</b> ( <b>\$newval</b> )
cpp	int <b>set_pulseDuration</b> ( double <b>newval</b> )
m	-(int) setPulseDuration : (double) <b>newval</b>
pas	function <b>set_pulseDuration</b> ( <b>newval</b> : double): integer
vb	function <b>set_pulseDuration</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_pulseDuration</b> ( double <b>newval</b> )
java	int <b>set_pulseDuration</b> ( double <b>newval</b> )
py	def <b>set_pulseDuration</b> ( <b>newval</b> )
cmd	YPwmOutput <b>target</b> <b>set_pulseDuration</b> <b>newval</b>

A pulse length cannot be longer than period, otherwise it is truncated.

#### Parameters :

**newval** a floating point number

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pwmoutput→set\_userdata()****YPwmOutput****pwmoutput→setUserData()****pwmoutput.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**pwmoutput→wait\_async()****YPwmOutput**

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.

## 3.31. PwmPowerSource function interface

The Yoctopuce application programming interface allows you to configure the voltage source used by all PWM on the same device.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_pwmpowersource.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPwmPowerSource = yoctolib.YPwmPowerSource;
php	require_once('yocto_pwmpowersource.php');
c++	#include "yocto_pwmpowersource.h"
m	#import "yocto_pwmpowersource.h"
pas	uses yocto_pwmpowersource;
vb	yocto_pwmpowersource.vb
cs	yocto_pwmpowersource.cs
java	import com.yoctopuce.YoctoAPI.YPwmPowerSource;
py	from yocto_pwmpowersource import *

### Global functions

#### yFindPwmPowerSource(func)

Retrieves a voltage source for a given identifier.

#### yFirstPwmPowerSource()

Starts the enumeration of Voltage sources currently accessible.

### YPwmPowerSource methods

#### pwmpowersource→describe()

Returns a short text that describes unambiguously the instance of the voltage source in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### pwmpowersource→get\_advertisedValue()

Returns the current value of the voltage source (no more than 6 characters).

#### pwmpowersource→get\_errorMessage()

Returns the error message of the latest error with the voltage source.

#### pwmpowersource→get\_errorType()

Returns the numerical error code of the latest error with the voltage source.

#### pwmpowersource→get\_friendlyName()

Returns a global identifier of the voltage source in the format MODULE\_NAME . FUNCTION\_NAME.

#### pwmpowersource→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### pwmpowersource→get\_functionId()

Returns the hardware identifier of the voltage source, without reference to the module.

#### pwmpowersource→get\_hardwareId()

Returns the unique hardware identifier of the voltage source in the form SERIAL . FUNCTIONID.

#### pwmpowersource→get\_logicalName()

Returns the logical name of the voltage source.

#### pwmpowersource→get\_module()

Gets the YModule object for the device on which the function is located.

#### pwmpowersource→get\_module\_async(callback, context)

Gets the YModule object for the device on which the function is located (asynchronous version).

**pwmpowersource→get\_powerMode()**

Returns the selected power source for the PWM on the same device

**pwmpowersource→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**pwmpowersource→isOnline()**

Checks if the voltage source is currently reachable, without raising any error.

**pwmpowersource→isOnline\_async(callback, context)**

Checks if the voltage source is currently reachable, without raising any error (asynchronous version).

**pwmpowersource→load(msValidity)**

Preloads the voltage source cache with a specified validity duration.

**pwmpowersource→load\_async(msValidity, callback, context)**

Preloads the voltage source cache with a specified validity duration (asynchronous version).

**pwmpowersource→nextPwmPowerSource()**

Continues the enumeration of Voltage sources started using yFirstPwmPowerSource( ).

**pwmpowersource→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**pwmpowersource→set\_logicalName(newval)**

Changes the logical name of the voltage source.

**pwmpowersource→set\_powerMode(newval)**

Changes the PWM power source.

**pwmpowersource→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**pwmpowersource→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.



## YPwmPowerSource.FindPwmPowerSource() yFindPwmPowerSource() YPwmPowerSource.FindPwmPowerSource()

## YPwmPowerSource

Retrieves a voltage source for a given identifier.

js	function <b>yFindPwmPowerSource</b> ( <b>func</b> )
nodejs	function <b>FindPwmPowerSource</b> ( <b>func</b> )
php	function <b>yFindPwmPowerSource</b> ( <b>\$func</b> )
cpp	YPwmPowerSource* <b>yFindPwmPowerSource</b> ( const string& <b>func</b> )
m	YPwmPowerSource* <b>yFindPwmPowerSource</b> ( NSString* <b>func</b> )
pas	function <b>yFindPwmPowerSource</b> ( <b>func</b> : string): TYPwmPowerSource
vb	function <b>yFindPwmPowerSource</b> ( ByVal <b>func</b> As String) As YPwmPowerSource
cs	YPwmPowerSource <b>FindPwmPowerSource</b> ( string <b>func</b> )
java	YPwmPowerSource <b>FindPwmPowerSource</b> ( String <b>func</b> )
py	def <b>FindPwmPowerSource</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the voltage source is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YPwmPowerSource.isOnline()` to test if the voltage source is indeed online at a given time. In case of ambiguity when looking for a voltage source 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 voltage source

### Returns :

a YPwmPowerSource object allowing you to drive the voltage source.

## YPwmPowerSource.FirstPwmPowerSource() yFirstPwmPowerSource() YPwmPowerSource.FirstPwmPowerSource()

YPwmPowerSource

Starts the enumeration of Voltage sources currently accessible.

js	function <b>yFirstPwmPowerSource</b> ( )
nodejs	function <b>FirstPwmPowerSource</b> ( )
php	function <b>yFirstPwmPowerSource</b> ( )
cpp	YPwmPowerSource* <b>yFirstPwmPowerSource</b> ( )
m	YPwmPowerSource* <b>yFirstPwmPowerSource</b> ( )
pas	function <b>yFirstPwmPowerSource</b> ( ): TYPwmPowerSource
vb	function <b>yFirstPwmPowerSource</b> ( ) As YPwmPowerSource
cs	YPwmPowerSource <b>FirstPwmPowerSource</b> ( )
java	YPwmPowerSource <b>FirstPwmPowerSource</b> ( )
py	def <b>FirstPwmPowerSource</b> ( )

Use the method `YPwmPowerSource.nextPwmPowerSource( )` to iterate on next Voltage sources.

### Returns :

a pointer to a `YPwmPowerSource` object, corresponding to the first source currently online, or a `null` pointer if there are none.

## pwmpowersource→describe() pwmpowersource.describe()

## YPwmPowerSource

Returns a short text that describes unambiguously the instance of the voltage source in the form  
`TYPE (NAME) = SERIAL . FUNCTIONID.`

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 voltage source (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pwmpowersource→get\_advertisedValue()

YPwmPowerSource

pwmpowersource→advertisedValue()

pwmpowersource.get\_advertisedValue()

Returns the current value of the voltage source (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YPwmPowerSource <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the voltage source (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**pwmpowersource→get\_errorMessage()**  
**pwmpowersource→errorMessage()**  
**pwmpowersource.get\_errorMessage()**

**YPwmPowerSource**

Returns the error message of the latest error with the voltage source.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the voltage source object

**pwmpowersource→get\_errorType()**  
**pwmpowersource→errorType()**  
**pwmpowersource.get\_errorType()**

**YPwmPowerSource**

Returns the numerical error code of the latest error with the voltage source.

<code>js</code>	<code>function get_errorType( )</code>
<code>nodejs</code>	<code>function get_errorType( )</code>
<code>php</code>	<code>function get_errorType( )</code>
<code>cpp</code>	<code>YRETCODE get_errorType( )</code>
<code>pas</code>	<code>function get_errorType( ): YRETCODE</code>
<code>vb</code>	<code>function get_errorType( ) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE get_errorType( )</code>
<code>java</code>	<code>int get_errorType( )</code>
<code>py</code>	<code>def get_errorType( )</code>

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 occurred while using the voltage source object

**pwmpowersource→get\_friendlyName()**  
**pwmpowersource→friendlyName()**  
**pwmpowersource.get\_friendlyName()**

**YPwmPowerSource**

Returns a global identifier of the voltage source in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the voltage source if they are defined, otherwise the serial number of the module and the hardware identifier of the voltage source (for example: `MyCustomName.relay1`)

#### Returns :

a string that uniquely identifies the voltage source using logical names (ex: `MyCustomName.relay1`)  
 On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**pwmpowersource→get\_functionDescriptor()**  
**pwmpowersource→functionDescriptor()**  
**pwmpowersource.get\_functionDescriptor()**

**YPwmPowerSource**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.



**pwmpowersource→get\_functionId()**  
**pwmpowersource→functionId()**  
**pwmpowersource.get\_functionId()**

**YPwmPowerSource**

Returns the hardware identifier of the voltage source, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the voltage source (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**pwmpowersource→get\_hardwareId()****YPwmPowerSource****pwmpowersource→hardwareId()****pwmpowersource.get\_hardwareId()**

Returns the unique hardware identifier of the voltage source in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the voltage source. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the voltage source (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**pwmpowersource→get\_logicalName()****YPwmPowerSource****pwmpowersource→logicalName()****pwmpowersource.get\_logicalName()**

Returns the logical name of the voltage source.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YPwmPowerSource <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the voltage source. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**pwmpowersource→get\_module()**  
**pwmpowersource→module()**  
**pwmpowersource.get\_module()**

**YPwmPowerSource**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**pwmpowersource→get\_module\_async()**  
**pwmpowersource→module\_async()**

---

**YPwmPowerSource**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**pwmpowersource**→**get\_powerMode()****YPwmPowerSource****pwmpowersource**→**powerMode()****pwmpowersource.get\_powerMode()**

Returns the selected power source for the PWM on the same device

js	function <b>get_powerMode</b> ( )
nodejs	function <b>get_powerMode</b> ( )
php	function <b>get_powerMode</b> ( )
cpp	Y_POWERMODE_enum <b>get_powerMode</b> ( )
m	-(Y_POWERMODE_enum) powerMode
pas	function <b>get_powerMode</b> ( ): Integer
vb	function <b>get_powerMode</b> ( ) As Integer
cs	int <b>get_powerMode</b> ( )
java	int <b>get_powerMode</b> ( )
py	def <b>get_powerMode</b> ( )

**Returns :**

a value among Y\_POWERMODE\_USB\_5V, Y\_POWERMODE\_USB\_3V, Y\_POWERMODE\_EXT\_V and Y\_POWERMODE\_OPNDRN corresponding to the selected power source for the PWM on the same device

On failure, throws an exception or returns Y\_POWERMODE\_INVALID.

**pwmpowersource→get\_userData()**  
**pwmpowersource→userData()**  
**pwmpowersource.get\_userData()**

**YPwmPowerSource**

Returns the value of the userData attribute, as previously stored using method set\_userData.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

## **pwmpowersource→isOnline()** **pwmpowersource.isOnline()**

**YPwmPowerSource**

Checks if the voltage source is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the voltage source 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 voltage source.

**Returns :**

`true` if the voltage source can be reached, and `false` otherwise



---

**pwmpowersource→isOnline\_async()****YPwmPowerSource**

---

Checks if the voltage source 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 voltage source 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.

**pwmpowersource**→**load()****pwmpowersource.load()****YPwmPowerSource**

Preloads the voltage source cache with a specified validity duration.

<code>js</code>	<code>function load( msValidity)</code>
<code>nodejs</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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 traffic 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.

**pwmpowersource→load\_async()****YPwmPowerSource**

Preloads the voltage source 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 traffic 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.

**pwmpowersource**→**nextPwmPowerSource()**  
**pwmpowersource.nextPwmPowerSource()**

**YPwmPowerSource**

Continues the enumeration of Voltage sources started using `yFirstPwmPowerSource()`.

js	function <b>nextPwmPowerSource</b> ( )
nodejs	function <b>nextPwmPowerSource</b> ( )
php	function <b>nextPwmPowerSource</b> ( )
c++	YPwmPowerSource * <b>nextPwmPowerSource</b> ( )
m	-(YPwmPowerSource*) <b>nextPwmPowerSource</b>
pas	function <b>nextPwmPowerSource</b> ( ): TYPwmPowerSource
vb	function <b>nextPwmPowerSource</b> ( ) As YPwmPowerSource
cs	YPwmPowerSource <b>nextPwmPowerSource</b> ( )
java	YPwmPowerSource <b>nextPwmPowerSource</b> ( )
py	def <b>nextPwmPowerSource</b> ( )

**Returns :**

a pointer to a `YPwmPowerSource` object, corresponding to a voltage source currently online, or a `null` pointer if there are no more Voltage sources to enumerate.

**pwmpowersource→registerValueCallback()**  
**pwmpowersource.registerValueCallback()**

**YPwmPowerSource**

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YPwmPowerSourceValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YPwmPowerSourceValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYPwmPowerSourceValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**pwmpowersource→set\_logicalName()**  
**pwmpowersource→setLogicalName()**  
**pwmpowersource.set\_logicalName()**

**YPwmPowerSource**

Changes the logical name of the voltage source.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YPwmPowerSource <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 voltage source.

#### Returns :

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**pwmpowersource→set\_powerMode()**  
**pwmpowersource→setPowerMode()**  
**pwmpowersource.set\_powerMode()**

**YPwmPowerSource**

Changes the PWM power source.

js	function <b>set_powerMode</b> ( <b>newval</b> )
nodejs	function <b>set_powerMode</b> ( <b>newval</b> )
php	function <b>set_powerMode</b> ( <b>\$newval</b> )
cpp	int <b>set_powerMode</b> ( Y_POWERMODE_enum <b>newval</b> )
m	-(int) setPowerMode : (Y_POWERMODE_enum) <b>newval</b>
pas	function <b>set_powerMode</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_powerMode</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_powerMode</b> ( int <b>newval</b> )
java	int <b>set_powerMode</b> ( int <b>newval</b> )
py	def <b>set_powerMode</b> ( <b>newval</b> )
cmd	YPwmPowerSource <b>target</b> <b>set_powerMode</b> <b>newval</b>

PWM can use isolated 5V from USB, isolated 3V from USB or voltage from an external power source. The PWM can also work in open drain mode. In that mode, the PWM actively pulls the line down. Warning: this setting is common to all PWM on the same device. If you change that parameter, all PWM located on the same device are affected. If you want the change to be kept after a device reboot, make sure to call the matching module `saveToFlash()`.

#### Parameters :

**newval** a value among Y\_POWERMODE\_USB\_5V, Y\_POWERMODE\_USB\_3V, Y\_POWERMODE\_EXT\_V and Y\_POWERMODE\_OPNDRN corresponding to the PWM power source

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**pwmpowersource→set\_userdata()****pwmpowersource→setUserData()****pwmpowersource.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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



---

**pwmpowersource→wait\_async()****YPwmPowerSource**

---

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.

## 3.32. Quaternion interface

The Yoctopuce API YQt class provides direct access to the Yocto3D attitude estimation using a quaternion. It is usually not needed to use the YQt class directly, as the YGyro class provides a more convenient higher-level interface.

In order to use the functions described here, you should include:

js	<code>&lt;script type='text/javascript' src='yocto_gyro.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YGyro = yoctolib.YGyro;</code>
php	<code>require_once('yocto_gyro.php');</code>
c++	<code>#include "yocto_gyro.h"</code>
m	<code>#import "yocto_gyro.h"</code>
pas	<code>uses yocto_gyro;</code>
vb	<code>yocto_gyro.vb</code>
cs	<code>yocto_gyro.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YGyro;</code>
py	<code>from yocto_gyro import *</code>

### Global functions

#### yFindQt(func)

Retrieves a quaternion component for a given identifier.

#### yFirstQt()

Starts the enumeration of quaternion components currently accessible.

### YQt methods

#### qt→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### qt→describe()

Returns a short text that describes unambiguously the instance of the quaternion component in the form TYPE ( NAME ) =SERIAL . FUNCTIONID.

#### qt→get\_advertisedValue()

Returns the current value of the quaternion component (no more than 6 characters).

#### qt→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### qt→get\_currentValue()

Returns the current value of the value.

#### qt→get\_errorMessage()

Returns the error message of the latest error with the quaternion component.

#### qt→get\_errorType()

Returns the numerical error code of the latest error with the quaternion component.

#### qt→get\_friendlyName()

Returns a global identifier of the quaternion component in the format MODULE\_NAME . FUNCTION\_NAME.

#### qt→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### qt→get\_functionId()

Returns the hardware identifier of the quaternion component, without reference to the module.

#### qt→get\_hardwareId()

Returns the unique hardware identifier of the quaternion component in the form `SERIAL.FUNCTIONID`.

#### **qt→get\_highestValue()**

Returns the maximal value observed for the value since the device was started.

#### **qt→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

#### **qt→get\_logicalName()**

Returns the logical name of the quaternion component.

#### **qt→get\_lowestValue()**

Returns the minimal value observed for the value since the device was started.

#### **qt→get\_module()**

Gets the `YModule` object for the device on which the function is located.

#### **qt→get\_module\_async(callback, context)**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

#### **qt→get\_recordedData(startTime, endTime)**

Retrieves a `DataSet` object holding historical data for this sensor, for a specified time interval.

#### **qt→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

#### **qt→get\_resolution()**

Returns the resolution of the measured values.

#### **qt→get\_unit()**

Returns the measuring unit for the value.

#### **qt→get\_userData()**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

#### **qt→isOnline()**

Checks if the quaternion component is currently reachable, without raising any error.

#### **qt→isOnline\_async(callback, context)**

Checks if the quaternion component is currently reachable, without raising any error (asynchronous version).

#### **qt→load(msValidity)**

Preloads the quaternion component cache with a specified validity duration.

#### **qt→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

#### **qt→load\_async(msValidity, callback, context)**

Preloads the quaternion component cache with a specified validity duration (asynchronous version).

#### **qt→nextQt()**

Continues the enumeration of quaternion components started using `yFirstQt()`.

#### **qt→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

#### **qt→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

#### **qt→set\_highestValue(newval)**

Changes the recorded maximal value observed.

#### **qt→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

#### **qt→set\_logicalName(newval)**

### 3. Reference

---

Changes the logical name of the quaternion component.

**qt**→**set\_lowestValue**(**newval**)

Changes the recorded minimal value observed.

**qt**→**set\_reportFrequency**(**newval**)

Changes the timed value notification frequency for this function.

**qt**→**set\_resolution**(**newval**)

Changes the resolution of the measured physical values.

**qt**→**set\_userData**(**data**)

Stores a user context provided as argument in the userData attribute of the function.

**qt**→**wait\_async**(**callback**, **context**)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YQt.FindQt()****YQt****yFindQt()YQt.FindQt()**

Retrieves a quaternion component for a given identifier.

js	function <b>yFindQt</b> ( <b>func</b> )
nodejs	function <b>FindQt</b> ( <b>func</b> )
php	function <b>yFindQt</b> ( <b>\$func</b> )
cpp	YQt* <b>yFindQt</b> ( string <b>func</b> )
m	+(YQt*) <b>yFindQt</b> : (NSString*) <b>func</b>
pas	function <b>yFindQt</b> ( <b>func</b> : string): TYQt
vb	function <b>yFindQt</b> ( ByVal <b>func</b> As String) As YQt
cs	YQt <b>FindQt</b> ( string <b>func</b> )
java	YQt <b>FindQt</b> ( String <b>func</b> )
py	def <b>FindQt</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the quaternion component is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YQt.isOnline()` to test if the quaternion component is indeed online at a given time. In case of ambiguity when looking for a quaternion component 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 quaternion component

**Returns :**

a `YQt` object allowing you to drive the quaternion component.

## YQt.FirstQt() yFirstQt()YQt.FirstQt()

YQt

Starts the enumeration of quaternion components currently accessible.

js	function <b>yFirstQt</b> ( )
nodejs	function <b>FirstQt</b> ( )
php	function <b>yFirstQt</b> ( )
cpp	YQt* <b>yFirstQt</b> ( )
m	YQt* <b>yFirstQt</b> ( )
pas	function <b>yFirstQt</b> ( ): TYQt
vb	function <b>yFirstQt</b> ( ) As YQt
cs	YQt <b>FirstQt</b> ( )
java	YQt <b>FirstQt</b> ( )
py	def <b>FirstQt</b> ( )

Use the method `YQt.nextQt( )` to iterate on next quaternion components.

### Returns :

a pointer to a `YQt` object, corresponding to the first quaternion component currently online, or a `null` pointer if there are none.

**qt→calibrateFromPoints()qt.calibrateFromPoints()****YQt**

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                             : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YSensor target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Parameters :**

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**qt→describe()qt.describe()****YQt**

Returns a short text that describes unambiguously the instance of the quaternion component in the form `TYPE (NAME) =SERIAL . FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 quaternion component (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)



**qt→get\_advertisedValue()****YQt****qt→advertisedValue()qt.get\_advertisedValue()**

Returns the current value of the quaternion component (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YSensor <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the quaternion component (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**qt→get\_currentRawValue()****YQt****qt→currentRawValue()qt.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YSensor <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**qt→get\_currentValue()****YQt****qt→currentValue()qt.get\_currentValue()**

Returns the current value of the value.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YSensor <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the value

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**qt→get\_errorMessage()****YQt****qt→errorMessage()qt.get\_errorMessage()**

Returns the error message of the latest error with the quaternion component.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the quaternion component object

**qt→get\_errorType()****YQt****qt→errorType()qt.get\_errorType()**

Returns the numerical error code of the latest error with the quaternion component.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the quaternion component object

**qt→get\_friendlyName()****YQt****qt→friendlyName()qt.get\_friendlyName()**

Returns a global identifier of the quaternion component in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the quaternion component if they are defined, otherwise the serial number of the module and the hardware identifier of the quaternion component (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the quaternion component using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**qt→get\_functionDescriptor()****YQt****qt→functionDescriptor()qt.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**qt→get\_functionId()****YQt****qt→functionId()qt.get\_functionId()**

Returns the hardware identifier of the quaternion component, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) functionId
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the quaternion component (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.



**qt→get\_hardwareId()****YQt****qt→hardwareId()qt.get\_hardwareId()**

Returns the unique hardware identifier of the quaternion component in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the quaternion component. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the quaternion component (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**qt→get\_highestValue()****YQt****qt→highestValue()qt.get\_highestValue()**

Returns the maximal value observed for the value since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YSensor <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the value since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**qt→get\_logFrequency()****YQt****qt→logFrequency()qt.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YSensor <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**qt→get\_logicalName()****YQt****qt→logicalName()****qt.get\_logicalName()**

Returns the logical name of the quaternion component.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YSensor <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the quaternion component. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**qt→get\_lowestValue()****qt→lowestValue()qt.get\_lowestValue()**

Returns the minimal value observed for the value since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YSensor <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the value since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**qt→get\_module()****YQt****qt→module()qt.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**qt→get\_module\_async()****YQt****qt→module\_async()**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**qt→get\_recordedData()****qt→recordedData()qt.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YSensor <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

#### Parameters :

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

#### Returns :

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.



**qt→get\_reportFrequency()****YQt****qt→reportFrequency()qt.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YSensor <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**qt→get\_resolution()****qt→resolution()qt.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YSensor <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**qt→get\_unit()****YQt****qt→unit()qt.get\_unit()**

Returns the measuring unit for the value.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YSensor <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the value

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**qt→get\_userdata()****YQt****qt→userData()qt.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**qt→isOnline()qt.isOnline()****YQt**

Checks if the quaternion component is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the quaternion component 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 quaternion component.

**Returns :**

`true` if the quaternion component can be reached, and `false` otherwise

**qt→isOnline\_async()****YQt**

Checks if the quaternion component 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 quaternion component 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.

**qt→load()qt.load()****YQt**

Preloads the quaternion component cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

**qt→loadCalibrationPoints()****qt.loadCalibrationPoints()****YQt**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js    function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php    function loadCalibrationPoints( &$rawValues, &$refValues)
cpp    int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m      -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas    function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )
cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)
cmd      YSensor target loadCalibrationPoints rawValues refValues

```

**Parameters :**

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**qt→load\_async()****YQt**

Preloads the quaternion component 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 traffic 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.

**qt→nextQt()qt.nextQt()****YQt**

Continues the enumeration of quaternion components started using `yFirstQt()`.

js	function <b>nextQt</b> ( )
nodejs	function <b>nextQt</b> ( )
php	function <b>nextQt</b> ( )
cpp	YQt * <b>nextQt</b> ( )
m	-(YQt*) <b>nextQt</b>
pas	function <b>nextQt</b> ( ): TYQt
vb	function <b>nextQt</b> ( ) As YQt
cs	YQt <b>nextQt</b> ( )
java	YQt <b>nextQt</b> ( )
py	def <b>nextQt</b> ( )

**Returns :**

a pointer to a `YQt` object, corresponding to a quaternion component currently online, or a `null` pointer if there are no more quaternion components to enumerate.

## qt→registerTimedReportCallback() qt.registerTimedReportCallback()

YQt

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YQtTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YQtTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYQtTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## qt→registerValueCallback() qt.registerValueCallback()

YQt

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YQtValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YQtValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYQtValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**qt→set\_highestValue()****YQt****qt→setHighestValue()qt.set\_highestValue()**

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YSensor <b>target</b> <b>set_highestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**qt→set\_logFrequency()****YQt****qt→setLogFrequency()qt.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YSensor <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**qt→set\_logicalName()****YQt****qt→setLogicalName()qt.set\_logicalName()**

Changes the logical name of the quaternion component.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YSensor <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 quaternion component.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**qt→set\_lowestValue()****YQt****qt→setLowestValue()qt.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YSensor <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**qt→set\_reportFrequency()****YQt****qt→setReportFrequency()qt.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YSensor <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**qt→set\_resolution()****qt→setResolution()qt.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YSensor <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**qt→set\_userdata()****YQt****qt→setUserData()qt.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**qt→wait\_async()**

YQt

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.

### 3.33. Real Time Clock function interface

The RealTimeClock function maintains and provides current date and time, even accross power cut lasting several days. It is the base for automated wake-up functions provided by the WakeUpScheduler. The current time may represent a local time as well as an UTC time, but no automatic time change will occur to account for daylight saving time.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_realtimeclock.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YRealTimeClock = yoctolib.YRealTimeClock;
php	require_once('yocto_realtimeclock.php');
c++	#include "yocto_realtimeclock.h"
m	#import "yocto_realtimeclock.h"
pas	uses yocto_realtimeclock;
vb	yocto_realtimeclock.vb
cs	yocto_realtimeclock.cs
java	import com.yoctopuce.YoctoAPI.YRealTimeClock;
py	from yocto_realtimeclock import *

#### Global functions

##### yFindRealTimeClock(func)

Retrieves a clock for a given identifier.

##### yFirstRealTimeClock()

Starts the enumeration of clocks currently accessible.

#### YRealTimeClock methods

##### realtimeclock→describe()

Returns a short text that describes unambiguously the instance of the clock in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

##### realtimeclock→get\_advertisedValue()

Returns the current value of the clock (no more than 6 characters).

##### realtimeclock→get\_dateTime()

Returns the current time in the form "YYYY/MM/DD hh:mm:ss"

##### realtimeclock→get\_errorMessage()

Returns the error message of the latest error with the clock.

##### realtimeclock→get\_errorType()

Returns the numerical error code of the latest error with the clock.

##### realtimeclock→get\_friendlyName()

Returns a global identifier of the clock in the format MODULE\_NAME . FUNCTION\_NAME.

##### realtimeclock→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

##### realtimeclock→get\_functionId()

Returns the hardware identifier of the clock, without reference to the module.

##### realtimeclock→get\_hardwareId()

Returns the unique hardware identifier of the clock in the form SERIAL . FUNCTIONID.

##### realtimeclock→get\_logicalName()

Returns the logical name of the clock.

##### realtimeclock→get\_module()

	Gets the <code>YModule</code> object for the device on which the function is located.
<b><code>realtimeclock→get_module_async(callback, context)</code></b>	Gets the <code>YModule</code> object for the device on which the function is located (asynchronous version).
<b><code>realtimeclock→get_timeSet()</code></b>	Returns true if the clock has been set, and false otherwise.
<b><code>realtimeclock→get_unixTime()</code></b>	Returns the current time in Unix format (number of elapsed seconds since Jan 1st, 1970).
<b><code>realtimeclock→get_userData()</code></b>	Returns the value of the <code>userData</code> attribute, as previously stored using method <code>set_userData</code> .
<b><code>realtimeclock→get_utcOffset()</code></b>	Returns the number of seconds between current time and UTC time (time zone).
<b><code>realtimeclock→isOnline()</code></b>	Checks if the clock is currently reachable, without raising any error.
<b><code>realtimeclock→isOnline_async(callback, context)</code></b>	Checks if the clock is currently reachable, without raising any error (asynchronous version).
<b><code>realtimeclock→load(msValidity)</code></b>	Preloads the clock cache with a specified validity duration.
<b><code>realtimeclock→load_async(msValidity, callback, context)</code></b>	Preloads the clock cache with a specified validity duration (asynchronous version).
<b><code>realtimeclock→nextRealTimeClock()</code></b>	Continues the enumeration of clocks started using <code>yFirstRealTimeClock()</code> .
<b><code>realtimeclock→registerValueCallback(callback)</code></b>	Registers the callback function that is invoked on every change of advertised value.
<b><code>realtimeclock→set_logicalName(newval)</code></b>	Changes the logical name of the clock.
<b><code>realtimeclock→set_unixTime(newval)</code></b>	Changes the current time.
<b><code>realtimeclock→set_userData(data)</code></b>	Stores a user context provided as argument in the <code>userData</code> attribute of the function.
<b><code>realtimeclock→set_utcOffset(newval)</code></b>	Changes the number of seconds between current time and UTC time (time zone).
<b><code>realtimeclock→wait_async(callback, context)</code></b>	Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YRealTimeClock.FindRealTimeClock() yFindRealTimeClock() YRealTimeClock.FindRealTimeClock()

## YRealTimeClock

Retrieves a clock for a given identifier.

js	function <b>yFindRealTimeClock</b> ( <b>func</b> )
nodejs	function <b>FindRealTimeClock</b> ( <b>func</b> )
php	function <b>yFindRealTimeClock</b> ( <b>\$func</b> )
cpp	YRealTimeClock* <b>yFindRealTimeClock</b> ( const string& <b>func</b> )
m	YRealTimeClock* <b>yFindRealTimeClock</b> ( NSString* <b>func</b> )
pas	function <b>yFindRealTimeClock</b> ( <b>func</b> : string): TYRealTimeClock
vb	function <b>yFindRealTimeClock</b> ( ByVal <b>func</b> As String) As YRealTimeClock
cs	YRealTimeClock <b>FindRealTimeClock</b> ( string <b>func</b> )
java	YRealTimeClock <b>FindRealTimeClock</b> ( String <b>func</b> )
py	def <b>FindRealTimeClock</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the clock is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YRealTimeClock.isOnline()` to test if the clock is indeed online at a given time. In case of ambiguity when looking for a clock 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 clock

### Returns :

a `YRealTimeClock` object allowing you to drive the clock.

## **YRealTimeClock.FirstRealTimeClock()** **yFirstRealTimeClock()** **YRealTimeClock.FirstRealTimeClock()**

**YRealTimeClock**

Starts the enumeration of clocks currently accessible.

js	function <b>yFirstRealTimeClock</b> ( )
nodejs	function <b>FirstRealTimeClock</b> ( )
php	function <b>yFirstRealTimeClock</b> ( )
cpp	YRealTimeClock* <b>yFirstRealTimeClock</b> ( )
m	YRealTimeClock* <b>yFirstRealTimeClock</b> ( )
pas	function <b>yFirstRealTimeClock</b> ( ): TYRealTimeClock
vb	function <b>yFirstRealTimeClock</b> ( ) As YRealTimeClock
cs	YRealTimeClock <b>FirstRealTimeClock</b> ( )
java	YRealTimeClock <b>FirstRealTimeClock</b> ( )
py	def <b>FirstRealTimeClock</b> ( )

Use the method `YRealTimeClock.nextRealTimeClock( )` to iterate on next clocks.

**Returns :**

a pointer to a `YRealTimeClock` object, corresponding to the first clock currently online, or a `null` pointer if there are none.



**realtimeclock→describe()realtimeclock.describe()****YRealTimeClock**

Returns a short text that describes unambiguously the instance of the clock in the form `TYPE (NAME) = SERIAL.FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 clock (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**realtimeclock→get\_advertisedValue()****YRealTimeClock****realtimeclock→advertisedValue()****realtimeclock.get\_advertisedValue()**

Returns the current value of the clock (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YRealTimeClock <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the clock (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

---

**realtimeclock→get\_dateTime()****YRealTimeClock****realtimeclock→dateTime()****realtimeclock.get\_dateTime()**

---

Returns the current time in the form "YYYY/MM/DD hh:mm:ss"

js	function <b>get_dateTime</b> ( )
nodejs	function <b>get_dateTime</b> ( )
php	function <b>get_dateTime</b> ( )
cpp	string <b>get_dateTime</b> ( )
m	-(NSString*) <b>dateTime</b>
pas	function <b>get_dateTime</b> ( ): string
vb	function <b>get_dateTime</b> ( ) As String
cs	string <b>get_dateTime</b> ( )
java	String <b>get_dateTime</b> ( )
py	def <b>get_dateTime</b> ( )

**Returns :**

a string corresponding to the current time in the form "YYYY/MM/DD hh:mm:ss"

On failure, throws an exception or returns Y\_DATETIME\_INVALID.

**realtimeclock→get\_errorMessage()****YRealTimeClock****realtimeclock→errorMessage()****realtimeclock.get\_errorMessage()**

Returns the error message of the latest error with the clock.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the clock object

**realtimeclock→get\_errorType()**  
**realtimeclock→errorType()**  
**realtimeclock.get\_errorType()**

**YRealTimeClock**

Returns the numerical error code of the latest error with the clock.

<code>js</code>	<code>function get_errorType( )</code>
<code>nodejs</code>	<code>function get_errorType( )</code>
<code>php</code>	<code>function get_errorType( )</code>
<code>cpp</code>	<code>YRETCODE get_errorType( )</code>
<code>pas</code>	<code>function get_errorType( ): YRETCODE</code>
<code>vb</code>	<code>function get_errorType( ) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE get_errorType( )</code>
<code>java</code>	<code>int get_errorType( )</code>
<code>py</code>	<code>def get_errorType( )</code>

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 occurred while using the clock object

**realtimeclock→get\_friendlyName()**  
**realtimeclock→friendlyName()**  
**realtimeclock.get\_friendlyName()**

**YRealTimeClock**

Returns a global identifier of the clock in the format `MODULE_NAME . FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the clock if they are defined, otherwise the serial number of the module and the hardware identifier of the clock (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the clock using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**realtimeclock→get\_functionDescriptor()**  
**realtimeclock→functionDescriptor()**  
**realtimeclock.get\_functionDescriptor()**

**YRealTimeClock**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**realtimeclock→get\_functionId()**  
**realtimeclock→functionId()**  
**realtimeclock.get\_functionId()**

**YRealTimeClock**

Returns the hardware identifier of the clock, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the clock (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.



**realtimeclock→get\_hardwareId()****YRealTimeClock****realtimeclock→hardwareId()****realtimeclock.get\_hardwareId()**

Returns the unique hardware identifier of the clock in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the clock. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the clock (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**realtimeclock→get\_logicalName()**  
**realtimeclock→logicalName()**  
**realtimeclock.get\_logicalName()**

**YRealTimeClock**

Returns the logical name of the clock.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YRealTimeClock <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the clock. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**realtimeclock→get\_module()****YRealTimeClock****realtimeclock→module()realtimeclock.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**realtimeclock→get\_module\_async()****YRealTimeClock****realtimeclock→module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**realtimeclock→get\_timeSet()****YRealTimeClock****realtimeclock→timeSet()realtimeclock.get\_timeSet()**

Returns true if the clock has been set, and false otherwise.

js	function <b>get_timeSet</b> ( )
nodejs	function <b>get_timeSet</b> ( )
php	function <b>get_timeSet</b> ( )
cpp	Y_TIMESET_enum <b>get_timeSet</b> ( )
m	-(Y_TIMESET_enum) timeSet
pas	function <b>get_timeSet</b> ( ): Integer
vb	function <b>get_timeSet</b> ( ) As Integer
cs	int <b>get_timeSet</b> ( )
java	int <b>get_timeSet</b> ( )
py	def <b>get_timeSet</b> ( )
cmd	YRealTimeClock <b>target get_timeSet</b>

**Returns :**

either Y\_TIMESET\_FALSE or Y\_TIMESET\_TRUE, according to true if the clock has been set, and false otherwise

On failure, throws an exception or returns Y\_TIMESET\_INVALID.

**realtimeclock→get\_unixTime()****YRealTimeClock****realtimeclock→unixTime()****realtimeclock.get\_unixTime()**

Returns the current time in Unix format (number of elapsed seconds since Jan 1st, 1970).

js	function <b>get_unixTime</b> ( )
nodejs	function <b>get_unixTime</b> ( )
php	function <b>get_unixTime</b> ( )
cpp	s64 <b>get_unixTime</b> ( )
m	-(s64) unixTime
pas	function <b>get_unixTime</b> ( ): int64
vb	function <b>get_unixTime</b> ( ) As Long
cs	long <b>get_unixTime</b> ( )
java	long <b>get_unixTime</b> ( )
py	def <b>get_unixTime</b> ( )
cmd	YRealTimeClock <b>target</b> <b>get_unixTime</b>

**Returns :**

an integer corresponding to the current time in Unix format (number of elapsed seconds since Jan 1st, 1970)

On failure, throws an exception or returns Y\_UNIXTIME\_INVALID.

**realtimeclock→get\_userdata()****YRealTimeClock****realtimeclock→userdata()****realtimeclock.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**realtimeclock→get\_utcOffset()****YRealTimeClock****realtimeclock→utcOffset()****realtimeclock.get\_utcOffset()**

Returns the number of seconds between current time and UTC time (time zone).

js	function <b>get_utcOffset</b> ( )
nodejs	function <b>get_utcOffset</b> ( )
php	function <b>get_utcOffset</b> ( )
cpp	int <b>get_utcOffset</b> ( )
m	-(int) utcOffset
pas	function <b>get_utcOffset</b> ( ): LongInt
vb	function <b>get_utcOffset</b> ( ) As Integer
cs	int <b>get_utcOffset</b> ( )
java	int <b>get_utcOffset</b> ( )
py	def <b>get_utcOffset</b> ( )
cmd	YRealTimeClock <b>target</b> <b>get_utcOffset</b>

**Returns :**

an integer corresponding to the number of seconds between current time and UTC time (time zone)

On failure, throws an exception or returns Y\_UTCOffset\_INVALID.



**realtimeclock→isOnline()realtimeclock.isOnline()****YRealTimeClock**

Checks if the clock is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the clock 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 clock.

**Returns :**

`true` if the clock can be reached, and `false` otherwise

**realtimeclock→isOnline\_async()****YRealTimeClock**

Checks if the clock 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 clock 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.

**realtimeclock→load()realtimeclock.load()****YRealTimeClock**

Preloads the clock cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

**realtimeclock→load\_async()****YRealTimeClock**

Preloads the clock 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 traffic 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.

## realtimeclock→nextRealTimeClock() realtimeclock.nextRealTimeClock()

## YRealTimeClock

Continues the enumeration of clocks started using `yFirstRealTimeClock()`.

js	function <b>nextRealTimeClock</b> ( )
nodejs	function <b>nextRealTimeClock</b> ( )
php	function <b>nextRealTimeClock</b> ( )
cpp	YRealTimeClock * <b>nextRealTimeClock</b> ( )
m	-(YRealTimeClock*) <b>nextRealTimeClock</b>
pas	function <b>nextRealTimeClock</b> ( ): TYRealTimeClock
vb	function <b>nextRealTimeClock</b> ( ) As YRealTimeClock
cs	YRealTimeClock <b>nextRealTimeClock</b> ( )
java	YRealTimeClock <b>nextRealTimeClock</b> ( )
py	def <b>nextRealTimeClock</b> ( )

### Returns :

a pointer to a `YRealTimeClock` object, corresponding to a clock currently online, or a `null` pointer if there are no more clocks to enumerate.

## realtimeclock→registerValueCallback() realtimeclock.registerValueCallback()

YRealTimeClock

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YRealTimeClockValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YRealTimeClockValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYRealTimeClockValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**realtimeclock→set\_logicalName()**  
**realtimeclock→setLogicalName()**  
**realtimeclock.set\_logicalName()**

**YRealTimeClock**

Changes the logical name of the clock.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YRealTimeClock <b>target set_logicalName newval</b>

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 clock.

#### Returns :

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**realtimeclock→set\_unixTime()**  
**realtimeclock→setUnixTime()**  
**realtimeclock.set\_unixTime()**

**YRealTimeClock**

Changes the current time.

js	function <b>set_unixTime</b> ( <b>newval</b> )
nodejs	function <b>set_unixTime</b> ( <b>newval</b> )
php	function <b>set_unixTime</b> ( <b>\$newval</b> )
cpp	int <b>set_unixTime</b> ( s64 <b>newval</b> )
m	-(int) setUnixTime : (s64) <b>newval</b>
pas	function <b>set_unixTime</b> ( <b>newval</b> : int64): integer
vb	function <b>set_unixTime</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_unixTime</b> ( long <b>newval</b> )
java	int <b>set_unixTime</b> ( long <b>newval</b> )
py	def <b>set_unixTime</b> ( <b>newval</b> )
cmd	YRealTimeClock <b>target</b> <b>set_unixTime</b> <b>newval</b>

Time is specifid in Unix format (number of elapsed seconds since Jan 1st, 1970). If current UTC time is known, utcOffset will be automatically adjusted for the new specified time.

**Parameters :**

**newval** an integer corresponding to the current time

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**realtimeclock→set\_userdata()**  
**realtimeclock→setUserData()**  
**realtimeclock.set\_userdata()**

**YRealTimeClock**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**realtimeclock→set\_utcOffset()**  
**realtimeclock→setUtcOffset()**  
**realtimeclock.set\_utcOffset()**

**YRealTimeClock**

Changes the number of seconds between current time and UTC time (time zone).

js	function <b>set_utcOffset</b> ( <b>newval</b> )
nodejs	function <b>set_utcOffset</b> ( <b>newval</b> )
php	function <b>set_utcOffset</b> ( <b>\$newval</b> )
cpp	int <b>set_utcOffset</b> ( int <b>newval</b> )
m	-(int) setUtcOffset : (int) <b>newval</b>
pas	function <b>set_utcOffset</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_utcOffset</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_utcOffset</b> ( int <b>newval</b> )
java	int <b>set_utcOffset</b> ( int <b>newval</b> )
py	def <b>set_utcOffset</b> ( <b>newval</b> )
cmd	YRealTimeClock <b>target set_utcOffset newval</b>

The timezone is automatically rounded to the nearest multiple of 15 minutes. If current UTC time is known, the current time will automatically be updated according to the selected time zone.

**Parameters :**

**newval** an integer corresponding to the number of seconds between current time and UTC time (time zone)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**realtimeclock→wait\_async()****YRealTimeClock**

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.

## 3.34. Reference frame configuration

This class is used to setup the base orientation of the Yocto-3D, so that the orientation functions, relative to the earth surface plane, use the proper reference frame. The class also implements a tridimensional sensor calibration process, which can compensate for local variations of standard gravity and improve the precision of the tilt sensors.

In order to use the functions described here, you should include:

js	<code>&lt;script type='text/javascript' src='yocto_refframe.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib'); var YRefFrame = yoctolib.YRefFrame;</code>
php	<code>require_once('yocto_refframe.php');</code>
c++	<code>#include "yocto_refframe.h"</code>
m	<code>#import "yocto_refframe.h"</code>
pas	<code>uses yocto_refframe;</code>
vb	<code>yocto_refframe.vb</code>
cs	<code>yocto_refframe.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YRefFrame;</code>
py	<code>from yocto_refframe import *</code>

### Global functions

#### **yFindRefFrame(func)**

Retrieves a reference frame for a given identifier.

#### **yFirstRefFrame()**

Starts the enumeration of reference frames currently accessible.

### YRefFrame methods

#### **refframe→cancel3DCalibration()**

Aborts the sensors tridimensional calibration process et restores normal settings.

#### **refframe→describe()**

Returns a short text that describes unambiguously the instance of the reference frame in the form `TYPE (NAME) = SERIAL.FUNCTIONID`.

#### **refframe→get\_3DCalibrationHint()**

Returns instructions to proceed to the tridimensional calibration initiated with method `start3DCalibration`.

#### **refframe→get\_3DCalibrationLogMsg()**

Returns the latest log message from the calibration process.

#### **refframe→get\_3DCalibrationProgress()**

Returns the global process indicator for the tridimensional calibration initiated with method `start3DCalibration`.

#### **refframe→get\_3DCalibrationStage()**

Returns index of the current stage of the calibration initiated with method `start3DCalibration`.

#### **refframe→get\_3DCalibrationStageProgress()**

Returns the process indicator for the current stage of the calibration initiated with method `start3DCalibration`.

#### **refframe→get\_advertisedValue()**

Returns the current value of the reference frame (no more than 6 characters).

#### **refframe→get\_bearing()**

Returns the reference bearing used by the compass.

**refframe→get\_errorMessage()**

Returns the error message of the latest error with the reference frame.

**refframe→get\_errorType()**

Returns the numerical error code of the latest error with the reference frame.

**refframe→get\_friendlyName()**

Returns a global identifier of the reference frame in the format `MODULE_NAME . FUNCTION_NAME`.

**refframe→get\_functionDescriptor()**

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

**refframe→get\_functionId()**

Returns the hardware identifier of the reference frame, without reference to the module.

**refframe→get\_hardwareId()**

Returns the unique hardware identifier of the reference frame in the form `SERIAL . FUNCTIONID`.

**refframe→get\_logicalName()**

Returns the logical name of the reference frame.

**refframe→get\_module()**

Gets the `YModule` object for the device on which the function is located.

**refframe→get\_module\_async(callback, context)**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

**refframe→get\_mountOrientation()**

Returns the installation orientation of the device, as configured in order to define the reference frame for the compass and the pitch/roll tilt sensors.

**refframe→get\_mountPosition()**

Returns the installation position of the device, as configured in order to define the reference frame for the compass and the pitch/roll tilt sensors.

**refframe→get\_userData()**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

**refframe→isOnline()**

Checks if the reference frame is currently reachable, without raising any error.

**refframe→isOnline\_async(callback, context)**

Checks if the reference frame is currently reachable, without raising any error (asynchronous version).

**refframe→load(msValidity)**

Preloads the reference frame cache with a specified validity duration.

**refframe→load\_async(msValidity, callback, context)**

Preloads the reference frame cache with a specified validity duration (asynchronous version).

**refframe→more3DCalibration()**

Continues the sensors tridimensional calibration process previously initiated using method `start3DCalibration`.

**refframe→nextRefFrame()**

Continues the enumeration of reference frames started using `yFirstRefFrame()`.

**refframe→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**refframe→save3DCalibration()**

Applies the sensors tridimensional calibration parameters that have just been computed.

**refframe→set\_bearing(newval)**

Changes the reference bearing used by the compass.

**refframe→set\_logicalName(newval)**

### 3. Reference

---

Changes the logical name of the reference frame.

**refframe→set\_mountPosition(position, orientation)**

Changes the compass and tilt sensor frame of reference.

**refframe→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**refframe→start3DCalibration()**

Initiates the sensors tridimensional calibration process.

**refframe→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YRefFrame.FindRefFrame() yFindRefFrame()YRefFrame.FindRefFrame()

## YRefFrame

Retrieves a reference frame for a given identifier.

js	function <b>yFindRefFrame</b> ( <b>func</b> )
nodejs	function <b>FindRefFrame</b> ( <b>func</b> )
php	function <b>yFindRefFrame</b> ( <b>\$func</b> )
cpp	YRefFrame* <b>yFindRefFrame</b> ( const string& <b>func</b> )
m	YRefFrame* <b>yFindRefFrame</b> ( NSString* <b>func</b> )
pas	function <b>yFindRefFrame</b> ( <b>func</b> : string): TYRefFrame
vb	function <b>yFindRefFrame</b> ( ByVal <b>func</b> As String) As YRefFrame
cs	YRefFrame <b>FindRefFrame</b> ( string <b>func</b> )
java	YRefFrame <b>FindRefFrame</b> ( String <b>func</b> )
py	def <b>FindRefFrame</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the reference frame is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YRefFrame.isOnline()` to test if the reference frame is indeed online at a given time. In case of ambiguity when looking for a reference frame 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 reference frame

### Returns :

a `YRefFrame` object allowing you to drive the reference frame.

**YRefFrame.FirstRefFrame()****YRefFrame****yFirstRefFrame()YRefFrame.FirstRefFrame()**

Starts the enumeration of reference frames currently accessible.

js	function <b>yFirstRefFrame</b> ( )
nodejs	function <b>FirstRefFrame</b> ( )
php	function <b>yFirstRefFrame</b> ( )
cpp	YRefFrame* <b>yFirstRefFrame</b> ( )
m	YRefFrame* <b>yFirstRefFrame</b> ( )
pas	function <b>yFirstRefFrame</b> ( ): TYRefFrame
vb	function <b>yFirstRefFrame</b> ( ) As YRefFrame
cs	YRefFrame <b>FirstRefFrame</b> ( )
java	YRefFrame <b>FirstRefFrame</b> ( )
py	def <b>FirstRefFrame</b> ( )

Use the method `YRefFrame.nextRefFrame( )` to iterate on next reference frames.

**Returns :**

a pointer to a `YRefFrame` object, corresponding to the first reference frame currently online, or a `null` pointer if there are none.



## refframe→cancel3DCalibration() refframe.cancel3DCalibration()

YRefFrame

Aborts the sensors tridimensional calibration process et restores normal settings.

js	function <b>cancel3DCalibration</b> ( )
nodejs	function <b>cancel3DCalibration</b> ( )
php	function <b>cancel3DCalibration</b> ( )
cpp	int <b>cancel3DCalibration</b> ( )
m	-(int) <b>cancel3DCalibration</b>
pas	function <b>cancel3DCalibration</b> ( ): LongInt
vb	function <b>cancel3DCalibration</b> ( ) As Integer
cs	int <b>cancel3DCalibration</b> ( )
java	int <b>cancel3DCalibration</b> ( )
py	def <b>cancel3DCalibration</b> ( )
cmd	YRefFrame <b>target</b> <b>cancel3DCalibration</b>

On failure, throws an exception or returns a negative error code.

**refframe→describe()****refframe.describe()****YRefFrame**

Returns a short text that describes unambiguously the instance of the reference frame in the form  
 TYPE ( NAME ) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 reference frame (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**refframe→get\_3DCalibrationHint()****YRefFrame****refframe→3DCalibrationHint()****refframe.get\_3DCalibrationHint()**

Returns instructions to proceed to the tridimensional calibration initiated with method `start3DCalibration`.

js	function <b>get_3DCalibrationHint</b> ( )
nodejs	function <b>get_3DCalibrationHint</b> ( )
php	function <b>get_3DCalibrationHint</b> ( )
cpp	string <b>get_3DCalibrationHint</b> ( )
m	-(NSString*) 3DCalibrationHint
pas	function <b>get_3DCalibrationHint</b> ( ): string
vb	function <b>get_3DCalibrationHint</b> ( ) As String
cs	string <b>get_3DCalibrationHint</b> ( )
java	String <b>get_3DCalibrationHint</b> ( )
py	def <b>get_3DCalibrationHint</b> ( )
cmd	YRefFrame <b>target</b> <b>get_3DCalibrationHint</b>

**Returns :**

a character string.

**refframe→get\_3DCalibrationLogMsg()****YRefFrame****refframe→3DCalibrationLogMsg()****refframe.get\_3DCalibrationLogMsg()**

Returns the latest log message from the calibration process.

js	function <b>get_3DCalibrationLogMsg</b> ( )
nodejs	function <b>get_3DCalibrationLogMsg</b> ( )
php	function <b>get_3DCalibrationLogMsg</b> ( )
cpp	string <b>get_3DCalibrationLogMsg</b> ( )
m	-(NSString*) 3DCalibrationLogMsg
pas	function <b>get_3DCalibrationLogMsg</b> ( ): string
vb	function <b>get_3DCalibrationLogMsg</b> ( ) As String
cs	string <b>get_3DCalibrationLogMsg</b> ( )
java	String <b>get_3DCalibrationLogMsg</b> ( )
py	def <b>get_3DCalibrationLogMsg</b> ( )
cmd	YRefFrame <b>target</b> <b>get_3DCalibrationLogMsg</b>

When no new message is available, returns an empty string.

**Returns :**

a character string.

**refframe→get\_3DCalibrationProgress()**  
**refframe→3DCalibrationProgress()**  
**refframe.get\_3DCalibrationProgress()**

**YRefFrame**

Returns the global process indicator for the tridimensional calibration initiated with method `start3DCalibration`.

js	function <b>get_3DCalibrationProgress</b> ( )
nodejs	function <b>get_3DCalibrationProgress</b> ( )
php	function <b>get_3DCalibrationProgress</b> ( )
cpp	int <b>get_3DCalibrationProgress</b> ( )
m	-(int) 3DCalibrationProgress
pas	function <b>get_3DCalibrationProgress</b> ( ): LongInt
vb	function <b>get_3DCalibrationProgress</b> ( ) As Integer
cs	int <b>get_3DCalibrationProgress</b> ( )
java	int <b>get_3DCalibrationProgress</b> ( )
py	def <b>get_3DCalibrationProgress</b> ( )
cmd	YRefFrame <b>target</b> <b>get_3DCalibrationProgress</b>

**Returns :**

an integer between 0 (not started) and 100 (stage completed).

**refframe→get\_3DCalibrationStage()****YRefFrame****refframe→3DCalibrationStage()****refframe.get\_3DCalibrationStage()**

Returns index of the current stage of the calibration initiated with method `start3DCalibration`.

js	function <b>get_3DCalibrationStage</b> ( )
nodejs	function <b>get_3DCalibrationStage</b> ( )
php	function <b>get_3DCalibrationStage</b> ( )
cpp	int <b>get_3DCalibrationStage</b> ( )
m	-(int) 3DCalibrationStage
pas	function <b>get_3DCalibrationStage</b> ( ): LongInt
vb	function <b>get_3DCalibrationStage</b> ( ) As Integer
cs	int <b>get_3DCalibrationStage</b> ( )
java	int <b>get_3DCalibrationStage</b> ( )
py	def <b>get_3DCalibrationStage</b> ( )
cmd	YRefFrame <b>target</b> <b>get_3DCalibrationStage</b>

**Returns :**

an integer, growing each time a calibration stage is completed.

**refframe→get\_3DCalibrationStageProgress()****YRefFrame****refframe→3DCalibrationStageProgress()****refframe.get\_3DCalibrationStageProgress()**

Returns the process indicator for the current stage of the calibration initiated with method `start3DCalibration`.

js	function <b>get_3DCalibrationStageProgress</b> ( )
nodejs	function <b>get_3DCalibrationStageProgress</b> ( )
php	function <b>get_3DCalibrationStageProgress</b> ( )
cpp	int <b>get_3DCalibrationStageProgress</b> ( )
m	-(int) 3DCalibrationStageProgress
pas	function <b>get_3DCalibrationStageProgress</b> ( ): LongInt
vb	function <b>get_3DCalibrationStageProgress</b> ( ) As Integer
cs	int <b>get_3DCalibrationStageProgress</b> ( )
java	int <b>get_3DCalibrationStageProgress</b> ( )
py	def <b>get_3DCalibrationStageProgress</b> ( )
cmd	YRefFrame <b>target</b> <b>get_3DCalibrationStageProgress</b>

**Returns :**

an integer between 0 (not started) and 100 (stage completed).

**refframe**→**get\_advertisedValue()****YRefFrame****refframe**→**advertisedValue()****refframe.get\_advertisedValue()**

Returns the current value of the reference frame (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YRefFrame <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the reference frame (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.



**refframe**→**get\_bearing()****YRefFrame****refframe**→**bearing()****refframe.get\_bearing()**

Returns the reference bearing used by the compass.

js	function <b>get_bearing</b> ( )
nodejs	function <b>get_bearing</b> ( )
php	function <b>get_bearing</b> ( )
cpp	double <b>get_bearing</b> ( )
m	-(double) bearing
pas	function <b>get_bearing</b> ( ): double
vb	function <b>get_bearing</b> ( ) As Double
cs	double <b>get_bearing</b> ( )
java	double <b>get_bearing</b> ( )
py	def <b>get_bearing</b> ( )
cmd	YRefFrame <b>target</b> <b>get_bearing</b>

The relative bearing indicated by the compass is the difference between the measured magnetic heading and the reference bearing indicated here.

**Returns :**

a floating point number corresponding to the reference bearing used by the compass

On failure, throws an exception or returns Y\_BEARING\_INVALID.

**refframe→get\_errorMessage()****YRefFrame****refframe→errorMessage()****refframe.get\_errorMessage()**

Returns the error message of the latest error with the reference frame.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the reference frame object

**refframe**→**get\_errorType()****YRefFrame****refframe**→**errorType()****refframe.get\_errorType()**

Returns the numerical error code of the latest error with the reference frame.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the reference frame object

**refframe**→**get\_friendlyName()****YRefFrame****refframe**→**friendlyName()****refframe.get\_friendlyName()**

Returns a global identifier of the reference frame in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
c++	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the reference frame if they are defined, otherwise the serial number of the module and the hardware identifier of the reference frame (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the reference frame using logical names (ex: `MyCustomName.relay1`)

On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**refframe→get\_functionDescriptor()****YRefFrame****refframe→functionDescriptor()****refframe.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**refframe**→**get\_functionId()****YRefFrame****refframe**→**functionId()****refframe.get\_functionId()**

Returns the hardware identifier of the reference frame, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the reference frame (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**refframe→get\_hardwareId()****YRefFrame****refframe→hardwareId()refframe.get\_hardwareId()**

Returns the unique hardware identifier of the reference frame in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the reference frame. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the reference frame (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**refframe→get\_logicalName()****YRefFrame****refframe→logicalName()****refframe.get\_logicalName()**

Returns the logical name of the reference frame.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YRefFrame <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the reference frame. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**refframe→get\_module()****YRefFrame****refframe→module()refframe.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**refframe→get\_module\_async()****YRefFrame****refframe→module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**refframe→get\_mountOrientation()****YRefFrame****refframe→mountOrientation()****refframe.get\_mountOrientation()**

Returns the installation orientation of the device, as configured in order to define the reference frame for the compass and the pitch/roll tilt sensors.

js	function <b>get_mountOrientation</b> ( )
nodejs	function <b>get_mountOrientation</b> ( )
php	function <b>get_mountOrientation</b> ( )
cpp	<b>Y_MOUNTORIENTATION</b> <b>get_mountOrientation</b> ( )
m	-( <b>Y_MOUNTORIENTATION</b> ) mountOrientation
pas	function <b>get_mountOrientation</b> ( ): <b>TY_MOUNTORIENTATION</b>
vb	function <b>get_mountOrientation</b> ( ) As <b>Y_MOUNTORIENTATION</b>
cs	<b>MOUNTORIENTATION</b> <b>get_mountOrientation</b> ( )
java	<b>MOUNTORIENTATION</b> <b>get_mountOrientation</b> ( )
py	def <b>get_mountOrientation</b> ( )
cmd	<b>YRefFrame</b> <b>target</b> <b>get_mountOrientation</b>

**Returns :**

a value among the enumeration **Y\_MOUNTORIENTATION** (**Y\_MOUNTORIENTATION\_TWELVE**, **Y\_MOUNTORIENTATION\_THREE**, **Y\_MOUNTORIENTATION\_SIX**, **Y\_MOUNTORIENTATION\_NINE**) corresponding to the orientation of the "X" arrow on the device, as on a clock dial seen from an observer in the center of the box. On the bottom face, the 12H orientation points to the front, while on the top face, the 12H orientation points to the rear.

On failure, throws an exception or returns a negative error code.

**refframe→get\_mountPosition()**
**YRefFrame**
**refframe→mountPosition()**
**refframe.get\_mountPosition()**

Returns the installation position of the device, as configured in order to define the reference frame for the compass and the pitch/roll tilt sensors.

js	function <b>get_mountPosition</b> ( )
nodejs	function <b>get_mountPosition</b> ( )
php	function <b>get_mountPosition</b> ( )
cpp	Y_MOUNTPOSITION <b>get_mountPosition</b> ( )
m	-(Y_MOUNTPOSITION) mountPosition
pas	function <b>get_mountPosition</b> ( ): TYMOUNTPOSITION
vb	function <b>get_mountPosition</b> ( ) As Y_MOUNTPOSITION
cs	MOUNTPOSITION <b>get_mountPosition</b> ( )
java	MOUNTPOSITION <b>get_mountPosition</b> ( )
py	def <b>get_mountPosition</b> ( )
cmd	YRefFrame <b>target</b> <b>get_mountPosition</b>

#### Returns :

a value among the Y\_MOUNTPOSITION enumeration (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), corresponding to the installation in a box, on one of the six faces.

On failure, throws an exception or returns a negative error code.

**refframe→get\_userdata()****YRefFrame****refframe→userdata()refframe.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method set\_userdata.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**refframe→isOnline()refframe.isOnline()****YRefFrame**

Checks if the reference frame is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the reference frame 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 reference frame.

**Returns :**

true if the reference frame can be reached, and false otherwise

**refframe→isOnline\_async()****YRefFrame**

Checks if the reference frame 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 reference frame 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.

## refframe→load()refframe.load()

## YRefFrame

Preloads the reference frame cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.



**refframe→load\_async()****YRefFrame**

Preloads the reference frame 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 traffic 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.

**refframe→more3DCalibration()**  
**refframe.more3DCalibration()****YRefFrame**

Continues the sensors tridimensional calibration process previously initiated using method `start3DCalibration`.

js	function <b>more3DCalibration</b> ( )
nodejs	function <b>more3DCalibration</b> ( )
php	function <b>more3DCalibration</b> ( )
cpp	int <b>more3DCalibration</b> ( )
m	-(int) <b>more3DCalibration</b>
pas	function <b>more3DCalibration</b> ( ): LongInt
vb	function <b>more3DCalibration</b> ( ) As Integer
cs	int <b>more3DCalibration</b> ( )
java	int <b>more3DCalibration</b> ( )
py	def <b>more3DCalibration</b> ( )
cmd	YRefFrame <b>target</b> <b>more3DCalibration</b>

This method should be called approximately 5 times per second, while positioning the device according to the instructions provided by method `get_3DCalibrationHint`. Note that the instructions change during the calibration process. On failure, throws an exception or returns a negative error code.

---

**refframe**→**nextRefFrame()****refframe.nextRefFrame()****YRefFrame**

---

Continues the enumeration of reference frames started using `yFirstRefFrame()`.

js	function <b>nextRefFrame</b> ( )
nodejs	function <b>nextRefFrame</b> ( )
php	function <b>nextRefFrame</b> ( )
cpp	YRefFrame * <b>nextRefFrame</b> ( )
m	-(YRefFrame*) <b>nextRefFrame</b>
pas	function <b>nextRefFrame</b> ( ): TYRefFrame
vb	function <b>nextRefFrame</b> ( ) As YRefFrame
cs	YRefFrame <b>nextRefFrame</b> ( )
java	YRefFrame <b>nextRefFrame</b> ( )
py	def <b>nextRefFrame</b> ( )

**Returns :**

a pointer to a `YRefFrame` object, corresponding to a reference frame currently online, or a `null` pointer if there are no more reference frames to enumerate.

## refframe→registerValueCallback() refframe.registerValueCallback()

YRefFrame

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YRefFrameValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YRefFrameValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYRefFrameValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

## refframe→save3DCalibration() refframe.save3DCalibration()

YRefFrame

Applies the sensors tridimensional calibration parameters that have just been computed.

js	function <b>save3DCalibration</b> ( )
nodejs	function <b>save3DCalibration</b> ( )
php	function <b>save3DCalibration</b> ( )
cpp	int <b>save3DCalibration</b> ( )
m	-(int) <b>save3DCalibration</b>
pas	function <b>save3DCalibration</b> ( ): LongInt
vb	function <b>save3DCalibration</b> ( ) As Integer
cs	int <b>save3DCalibration</b> ( )
java	int <b>save3DCalibration</b> ( )
py	def <b>save3DCalibration</b> ( )
cmd	YRefFrame <b>target</b> <b>save3DCalibration</b>

Remember to call the `saveToFlash()` method of the module if the changes must be kept when the device is restarted. On failure, throws an exception or returns a negative error code.

refframe→set\_bearing()

YRefFrame

refframe→setBearing()refframe.set\_bearing()

Changes the reference bearing used by the compass.

js	function set_bearing( newval)
nodejs	function set_bearing( newval)
php	function set_bearing( \$newval)
cpp	int set_bearing( double newval)
m	-(int) setBearing : (double) newval
pas	function set_bearing( newval: double): integer
vb	function set_bearing( ByVal newval As Double) As Integer
cs	int set_bearing( double newval)
java	int set_bearing( double newval)
py	def set_bearing( newval)
cmd	YRefFrame target set_bearing newval

The relative bearing indicated by the compass is the difference between the measured magnetic heading and the reference bearing indicated here. For instance, if you setup as reference bearing the value of the earth magnetic declination, the compass will provide the orientation relative to the geographic North. Similarly, when the sensor is not mounted along the standard directions because it has an additional yaw angle, you can set this angle in the reference bearing so that the compass provides the expected natural direction. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

#### Parameters :

**newval** a floating point number corresponding to the reference bearing used by the compass

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**refframe→set\_logicalName()****YRefFrame****refframe→setLogicalName()****refframe.set\_logicalName()**

Changes the logical name of the reference frame.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YRefFrame <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 reference frame.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

refframe→set\_mountPosition()  
 refframe→setMountPosition()  
 refframe.set\_mountPosition()

YRefFrame

Changes the compass and tilt sensor frame of reference.

```

js function set_mountPosition( position, orientation)
nodejs function set_mountPosition( position, orientation)
php function set_mountPosition( $position, $orientation)
cpp int set_mountPosition( Y_MOUNTPOSITION position,
                           Y_MOUNTORIENTATION orientation)

m -(int) setMountPosition : (Y_MOUNTPOSITION) position
                           : (Y_MOUNTORIENTATION) orientation

pas function set_mountPosition( position: TYMOUNTPOSITION,
                               orientation: TYMOUNTORIENTATION): LongInt

vb function set_mountPosition( ) As Integer
cs int set_mountPosition( MOUNTPOSITION position,
                          MOUNTORIENTATION orientation)
java int set_mountPosition( MOUNTPOSITION position,
                             MOUNTORIENTATION orientation)
py def set_mountPosition( position, orientation)
cmd YRefFrame target set_mountPosition position orientation

```

The magnetic compass and the tilt sensors (pitch and roll) naturally work in the plane parallel to the earth surface. In case the device is not installed upright and horizontally, you must select its reference orientation (parallel to the earth surface) so that the measures are made relative to this position.

#### Parameters :

- position** a value among the Y\_MOUNTPOSITION enumeration (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), corresponding to the installation in a box, on one of the six faces.
- orientation** a value among the enumeration Y\_MOUNTORIENTATION (Y\_MOUNTORIENTATION\_TWELVE, Y\_MOUNTORIENTATION\_THREE, Y\_MOUNTORIENTATION\_SIX, Y\_MOUNTORIENTATION\_NINE) corresponding to the orientation of the "X" arrow on the device, as on a clock dial seen from an observer in the center of the box. On the bottom face, the 12H orientation points to the front, while on the top face, the 12H orientation points to the rear. Remember to call the saveToFlash( ) method of the module if the modification must be kept.



**refframe→set\_userdata()****YRefFrame****refframe→setUserData()refframe.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
c++	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

## refframe→start3DCalibration() refframe.start3DCalibration()

YRefFrame

Initiates the sensors tridimensional calibration process.

js	function <b>start3DCalibration</b> ( )
nodejs	function <b>start3DCalibration</b> ( )
php	function <b>start3DCalibration</b> ( )
cpp	int <b>start3DCalibration</b> ( )
m	-(int) <b>start3DCalibration</b>
pas	function <b>start3DCalibration</b> ( ): LongInt
vb	function <b>start3DCalibration</b> ( ) As Integer
cs	int <b>start3DCalibration</b> ( )
java	int <b>start3DCalibration</b> ( )
py	def <b>start3DCalibration</b> ( )
cmd	YRefFrame <b>target start3DCalibration</b>

This calibration is used at low level for inertial position estimation and to enhance the precision of the tilt sensors. After calling this method, the device should be moved according to the instructions provided by method `get_3DCalibrationHint`, and `more3DCalibration` should be invoked about 5 times per second. The calibration procedure is completed when the method `get_3DCalibrationProgress` returns 100. At this point, the computed calibration parameters can be applied using method `save3DCalibration`. The calibration process can be canceled at any time using method `cancel3DCalibration`. On failure, throws an exception or returns a negative error code.

**refframe→wait\_async()****YRefFrame**

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.

## 3.35. Relay function interface

The Yoctopuce application programming interface allows you to switch the relay state. This change is not persistent: the relay will automatically return to its idle position whenever power is lost or if the module is restarted. The library can also generate automatically short pulses of determined duration. On devices with two output for each relay (double throw), the two outputs are named A and B, with output A corresponding to the idle position (at power off) and the output B corresponding to the active state. If you prefer the alternate default state, simply switch your cables on the board.

In order to use the functions described here, you should include:

js	<code>&lt;script type='text/javascript' src='yocto_relay.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YRelay = yoctolib.YRelay;</code>
php	<code>require_once('yocto_relay.php');</code>
c++	<code>#include "yocto_relay.h"</code>
m	<code>#import "yocto_relay.h"</code>
pas	<code>uses yocto_relay;</code>
vb	<code>yocto_relay.vb</code>
cs	<code>yocto_relay.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YRelay;</code>
py	<code>from yocto_relay import *</code>

### Global functions

#### yFindRelay(func)

Retrieves a relay for a given identifier.

#### yFirstRelay()

Starts the enumeration of relays currently accessible.

### YRelay methods

#### relay→delayedPulse(ms\_delay, ms\_duration)

Schedules a pulse.

#### relay→describe()

Returns a short text that describes unambiguously the instance of the relay in the form  
TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### relay→get\_advertisedValue()

Returns the current value of the relay (no more than 6 characters).

#### relay→get\_countdown()

Returns the number of milliseconds remaining before a pulse (delayedPulse() call) When there is no scheduled pulse, returns zero.

#### relay→get\_errorMessage()

Returns the error message of the latest error with the relay.

#### relay→get\_errorType()

Returns the numerical error code of the latest error with the relay.

#### relay→get\_friendlyName()

Returns a global identifier of the relay in the format MODULE\_NAME . FUNCTION\_NAME.

#### relay→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### relay→get\_functionId()

Returns the hardware identifier of the relay, without reference to the module.

#### relay→get\_hardwareId()

Returns the unique hardware identifier of the relay in the form `SERIAL.FUNCTIONID`.

**relay→get\_logicalName()**

Returns the logical name of the relay.

**relay→get\_maxTimeOnStateA()**

Retourne the maximum time (ms) allowed for `$THEFUNCTIONS$` to stay in state A before automatically switching back in to B state.

**relay→get\_maxTimeOnStateB()**

Retourne the maximum time (ms) allowed for `$THEFUNCTIONS$` to stay in state B before automatically switching back in to A state.

**relay→get\_module()**

Gets the `YModule` object for the device on which the function is located.

**relay→get\_module\_async(callback, context)**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

**relay→get\_output()**

Returns the output state of the relays, when used as a simple switch (single throw).

**relay→get\_pulseTimer()**

Returns the number of milliseconds remaining before the relays is returned to idle position (state A), during a measured pulse generation.

**relay→get\_state()**

Returns the state of the relays (A for the idle position, B for the active position).

**relay→get\_stateAtPowerOn()**

Returns the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no change).

**relay→get\_userData()**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

**relay→isOnline()**

Checks if the relay is currently reachable, without raising any error.

**relay→isOnline\_async(callback, context)**

Checks if the relay is currently reachable, without raising any error (asynchronous version).

**relay→load(msValidity)**

Preloads the relay cache with a specified validity duration.

**relay→load\_async(msValidity, callback, context)**

Preloads the relay cache with a specified validity duration (asynchronous version).

**relay→nextRelay()**

Continues the enumeration of relays started using `yFirstRelay()`.

**relay→pulse(ms\_duration)**

Sets the relay to output B (active) for a specified duration, then brings it automatically back to output A (idle state).

**relay→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**relay→set\_logicalName(newval)**

Changes the logical name of the relay.

**relay→set\_maxTimeOnStateA(newval)**

Sets the maximum time (ms) allowed for `$THEFUNCTIONS$` to stay in state A before automatically switching back in to B state.

**relay→set\_maxTimeOnStateB(newval)**

### 3. Reference

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

#### **relay→set\_output(newval)**

Changes the output state of the relays, when used as a simple switch (single throw).

#### **relay→set\_state(newval)**

Changes the state of the relays (A for the idle position, B for the active position).

#### **relay→set\_stateAtPowerOn(newval)**

Preset the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no modification).

#### **relay→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

#### **relay→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YRelay.FindRelay()****YRelay****yFindRelay()YRelay.FindRelay()**

Retrieves a relay for a given identifier.

<code>js</code>	<code>function yFindRelay( func)</code>
<code>nodejs</code>	<code>function FindRelay( func)</code>
<code>php</code>	<code>function yFindRelay( \$func)</code>
<code>cpp</code>	<code>YRelay* yFindRelay( const string&amp; func)</code>
<code>m</code>	<code>YRelay* yFindRelay( NSString* func)</code>
<code>pas</code>	<code>function yFindRelay( func: string): TYRelay</code>
<code>vb</code>	<code>function yFindRelay( ByVal func As String) As YRelay</code>
<code>cs</code>	<code>YRelay FindRelay( string func)</code>
<code>java</code>	<code>YRelay FindRelay( String func)</code>
<code>py</code>	<code>def FindRelay( func)</code>

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the relay is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YRelay.isOnline()` to test if the relay is indeed online at a given time. In case of ambiguity when looking for a relay 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 relay

**Returns :**

a `YRelay` object allowing you to drive the relay.

**YRelay.FirstRelay()****YRelay****yFirstRelay()YRelay.FirstRelay()**

Starts the enumeration of relays currently accessible.

js	function <b>yFirstRelay</b> ( )
nodejs	function <b>FirstRelay</b> ( )
php	function <b>yFirstRelay</b> ( )
cpp	YRelay* <b>yFirstRelay</b> ( )
m	YRelay* <b>yFirstRelay</b> ( )
pas	function <b>yFirstRelay</b> ( ): TYRelay
vb	function <b>yFirstRelay</b> ( ) As YRelay
cs	YRelay <b>FirstRelay</b> ( )
java	YRelay <b>FirstRelay</b> ( )
py	def <b>FirstRelay</b> ( )

Use the method `YRelay.nextRelay( )` to iterate on next relays.

**Returns :**

a pointer to a YRelay object, corresponding to the first relay currently online, or a `null` pointer if there are none.



**relay→delayedPulse()relay.delayedPulse()****YRelay**

Schedules a pulse.

js	function <b>delayedPulse</b> ( <b>ms_delay</b> , <b>ms_duration</b> )
nodejs	function <b>delayedPulse</b> ( <b>ms_delay</b> , <b>ms_duration</b> )
php	function <b>delayedPulse</b> ( <b>\$ms_delay</b> , <b>\$ms_duration</b> )
cpp	int <b>delayedPulse</b> ( int <b>ms_delay</b> , int <b>ms_duration</b> )
m	-(int) <b>delayedPulse</b> : (int) <b>ms_delay</b> : (int) <b>ms_duration</b>
pas	function <b>delayedPulse</b> ( <b>ms_delay</b> : LongInt, <b>ms_duration</b> : LongInt): integer
vb	function <b>delayedPulse</b> ( ByVal <b>ms_delay</b> As Integer, ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>delayedPulse</b> ( int <b>ms_delay</b> , int <b>ms_duration</b> )
java	int <b>delayedPulse</b> ( int <b>ms_delay</b> , int <b>ms_duration</b> )
py	def <b>delayedPulse</b> ( <b>ms_delay</b> , <b>ms_duration</b> )
cmd	YRelay <b>target</b> <b>delayedPulse</b> <b>ms_delay</b> <b>ms_duration</b>

**Parameters :****ms\_delay** waiting time before the pulse, in milliseconds**ms\_duration** pulse duration, in milliseconds**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**relay→describe()****relay.describe()****YRelay**

Returns a short text that describes unambiguously the instance of the relay in the form  
 TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 relay (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**relay**→**get\_advertisedValue()****YRelay****relay**→**advertisedValue()****relay.get\_advertisedValue()**

Returns the current value of the relay (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YRelay <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the relay (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**relay**→**get\_countdown()****YRelay****relay**→**countdown()****relay.get\_countdown()**

Returns the number of milliseconds remaining before a pulse (delayedPulse() call) When there is no scheduled pulse, returns zero.

js	function <b>get_countdown</b> ( )
nodejs	function <b>get_countdown</b> ( )
php	function <b>get_countdown</b> ( )
cpp	s64 <b>get_countdown</b> ( )
m	-(s64) countdown
pas	function <b>get_countdown</b> ( ): int64
vb	function <b>get_countdown</b> ( ) As Long
cs	long <b>get_countdown</b> ( )
java	long <b>get_countdown</b> ( )
py	def <b>get_countdown</b> ( )
cmd	YRelay <b>target</b> <b>get_countdown</b>

**Returns :**

an integer corresponding to the number of milliseconds remaining before a pulse (delayedPulse() call) When there is no scheduled pulse, returns zero

On failure, throws an exception or returns Y\_COUNTDOWN\_INVALID.

**relay**→**get\_errorMessage()****YRelay****relay**→**errorMessage()****relay.get\_errorMessage()**

Returns the error message of the latest error with the relay.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the relay object

**relay**→**get\_errorType()****YRelay****relay**→**errorType()****relay.get\_errorType()**

Returns the numerical error code of the latest error with the relay.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the relay object

**relay**→**get\_friendlyName()****YRelay****relay**→**friendlyName()****relay.get\_friendlyName()**

Returns a global identifier of the relay in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the relay if they are defined, otherwise the serial number of the module and the hardware identifier of the relay (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the relay using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**relay**→**get\_functionDescriptor()****YRelay****relay**→**functionDescriptor()****relay.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.



**relay**→**get\_functionId()****YRelay****relay**→**functionId()****relay.get\_functionId()**

Returns the hardware identifier of the relay, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the relay (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**relay**→**get\_hardwareId()****YRelay****relay**→**hardwareId()****relay.get\_hardwareId()**

Returns the unique hardware identifier of the relay in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the relay. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the relay (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**relay**→**get\_logicalName()****YRelay****relay**→**logicalName()****relay.get\_logicalName()**

Returns the logical name of the relay.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YRelay <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the relay. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**relay**→**get\_maxTimeOnStateA()****YRelay****relay**→**maxTimeOnStateA()****relay.get\_maxTimeOnStateA()**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

js	function <b>get_maxTimeOnStateA</b> ( )
nodejs	function <b>get_maxTimeOnStateA</b> ( )
php	function <b>get_maxTimeOnStateA</b> ( )
cpp	s64 <b>get_maxTimeOnStateA</b> ( )
m	-(s64) maxTimeOnStateA
pas	function <b>get_maxTimeOnStateA</b> ( ): int64
vb	function <b>get_maxTimeOnStateA</b> ( ) As Long
cs	long <b>get_maxTimeOnStateA</b> ( )
java	long <b>get_maxTimeOnStateA</b> ( )
py	def <b>get_maxTimeOnStateA</b> ( )
cmd	YRelay <b>target</b> <b>get_maxTimeOnStateA</b>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEA\_INVALID.

**relay→get\_maxTimeOnStateB()****YRelay****relay→maxTimeOnStateB()****relay.get\_maxTimeOnStateB()**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

js	function <b>get_maxTimeOnStateB</b> ( )
nodejs	function <b>get_maxTimeOnStateB</b> ( )
php	function <b>get_maxTimeOnStateB</b> ( )
cpp	s64 <b>get_maxTimeOnStateB</b> ( )
m	-(s64) maxTimeOnStateB
pas	function <b>get_maxTimeOnStateB</b> ( ): int64
vb	function <b>get_maxTimeOnStateB</b> ( ) As Long
cs	long <b>get_maxTimeOnStateB</b> ( )
java	long <b>get_maxTimeOnStateB</b> ( )
py	def <b>get_maxTimeOnStateB</b> ( )
cmd	YRelay <b>target</b> <b>get_maxTimeOnStateB</b>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEB\_INVALID.

**relay**→**get\_module()****YRelay****relay**→**module()****relay.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**relay→get\_module\_async()****YRelay****relay→module\_async()**

---

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**relay**→**get\_output()****YRelay****relay**→**output()****relay.get\_output()**

Returns the output state of the relays, when used as a simple switch (single throw).

js	function <b>get_output</b> ( )
nodejs	function <b>get_output</b> ( )
php	function <b>get_output</b> ( )
cpp	Y_OUTPUT_enum <b>get_output</b> ( )
m	-(Y_OUTPUT_enum) output
pas	function <b>get_output</b> ( ): Integer
vb	function <b>get_output</b> ( ) As Integer
cs	int <b>get_output</b> ( )
java	int <b>get_output</b> ( )
py	def <b>get_output</b> ( )
cmd	YRelay <b>target</b> <b>get_output</b>

**Returns :**

either Y\_OUTPUT\_OFF or Y\_OUTPUT\_ON, according to the output state of the relays, when used as a simple switch (single throw)

On failure, throws an exception or returns Y\_OUTPUT\_INVALID.



**relay**→**get\_pulseTimer()****YRelay****relay**→**pulseTimer()****relay.get\_pulseTimer()**

Returns the number of milliseconds remaining before the relays is returned to idle position (state A), during a measured pulse generation.

js	function <b>get_pulseTimer</b> ( )
nodejs	function <b>get_pulseTimer</b> ( )
php	function <b>get_pulseTimer</b> ( )
cpp	s64 <b>get_pulseTimer</b> ( )
m	-(s64) pulseTimer
pas	function <b>get_pulseTimer</b> ( ): int64
vb	function <b>get_pulseTimer</b> ( ) As Long
cs	long <b>get_pulseTimer</b> ( )
java	long <b>get_pulseTimer</b> ( )
py	def <b>get_pulseTimer</b> ( )
cmd	YRelay <b>target</b> <b>get_pulseTimer</b>

When there is no ongoing pulse, returns zero.

**Returns :**

an integer corresponding to the number of milliseconds remaining before the relays is returned to idle position (state A), during a measured pulse generation

On failure, throws an exception or returns Y\_PULSETIMER\_INVALID.

**relay**→**get\_state()****YRelay****relay**→**state()****relay.get\_state()**

Returns the state of the relays (A for the idle position, B for the active position).

js	function <b>get_state</b> ( )
nodejs	function <b>get_state</b> ( )
php	function <b>get_state</b> ( )
cpp	Y_STATE_enum <b>get_state</b> ( )
m	-(Y_STATE_enum) state
pas	function <b>get_state</b> ( ): Integer
vb	function <b>get_state</b> ( ) As Integer
cs	int <b>get_state</b> ( )
java	int <b>get_state</b> ( )
py	def <b>get_state</b> ( )
cmd	YRelay <b>target</b> <b>get_state</b>

**Returns :**

either Y\_STATE\_A or Y\_STATE\_B, according to the state of the relays (A for the idle position, B for the active position)

On failure, throws an exception or returns Y\_STATE\_INVALID.

**relay→get\_stateAtPowerOn()****YRelay****relay→stateAtPowerOn()relay.get\_stateAtPowerOn()**

Returns the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no change).

js	function <b>get_stateAtPowerOn</b> ( )
nodejs	function <b>get_stateAtPowerOn</b> ( )
php	function <b>get_stateAtPowerOn</b> ( )
cpp	Y_STATEATPOWERON_enum <b>get_stateAtPowerOn</b> ( )
m	-(Y_STATEATPOWERON_enum) stateAtPowerOn
pas	function <b>get_stateAtPowerOn</b> ( ): Integer
vb	function <b>get_stateAtPowerOn</b> ( ) As Integer
cs	int <b>get_stateAtPowerOn</b> ( )
java	int <b>get_stateAtPowerOn</b> ( )
py	def <b>get_stateAtPowerOn</b> ( )
cmd	YRelay <b>target</b> <b>get_stateAtPowerOn</b>

**Returns :**

a value among Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A and Y\_STATEATPOWERON\_B corresponding to the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no change)

On failure, throws an exception or returns Y\_STATEATPOWERON\_INVALID.

**relay**→**get\_userData()****YRelay****relay**→**userData()****relay.userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**relay→isOnline()relay.isOnline()****YRelay**

Checks if the relay is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the relay 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 relay.

**Returns :**

`true` if the relay can be reached, and `false` otherwise

**relay→isOnline\_async()****YRelay**

Checks if the relay 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 relay 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.

**relay→load()relay.load()****YRelay**

Preloads the relay cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

**relay→load\_async()****YRelay**

Preloads the relay 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 traffic 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.



**relay**→**nextRelay()****relay.nextRelay()****YRelay**

Continues the enumeration of relays started using `yFirstRelay()`.

<code>js</code>	<code>function nextRelay( )</code>
<code>nodejs</code>	<code>function nextRelay( )</code>
<code>php</code>	<code>function nextRelay( )</code>
<code>cpp</code>	<code>YRelay * nextRelay( )</code>
<code>m</code>	<code>-(YRelay*) nextRelay</code>
<code>pas</code>	<code>function nextRelay( ): TYRelay</code>
<code>vb</code>	<code>function nextRelay( ) As YRelay</code>
<code>cs</code>	<code>YRelay nextRelay( )</code>
<code>java</code>	<code>YRelay nextRelay( )</code>
<code>py</code>	<code>def nextRelay( )</code>

**Returns :**

a pointer to a `YRelay` object, corresponding to a relay currently online, or a `null` pointer if there are no more relays to enumerate.

**relay→pulse()relay.pulse()****YRelay**

Sets the relay to output B (active) for a specified duration, then brings it automatically back to output A (idle state).

js	function <b>pulse</b> ( <b>ms_duration</b> )
nodejs	function <b>pulse</b> ( <b>ms_duration</b> )
php	function <b>pulse</b> ( <b>\$ms_duration</b> )
cpp	int <b>pulse</b> ( int <b>ms_duration</b> )
m	-(int) <b>pulse</b> : (int) <b>ms_duration</b>
pas	function <b>pulse</b> ( <b>ms_duration</b> : LongInt): integer
vb	function <b>pulse</b> ( ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>pulse</b> ( int <b>ms_duration</b> )
java	int <b>pulse</b> ( int <b>ms_duration</b> )
py	def <b>pulse</b> ( <b>ms_duration</b> )
cmd	YRelay <b>target pulse ms_duration</b>

**Parameters :**

**ms\_duration** pulse duration, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

## relay→registerValueCallback() relay.registerValueCallback()

YRelay

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YRelayValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YRelayValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYRelayValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**relay**→**set\_logicalName()****YRelay****relay**→**setLogicalName()****relay.set\_logicalName()**

Changes the logical name of the relay.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YRelay <b>target set_logicalName newval</b>

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 relay.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**relay**→**set\_maxTimeOnStateA()****YRelay****relay**→**setMaxTimeOnStateA()****relay.set\_maxTimeOnStateA()**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

js	function <b>set_maxTimeOnStateA</b> ( <b>newval</b> )
nodejs	function <b>set_maxTimeOnStateA</b> ( <b>newval</b> )
php	function <b>set_maxTimeOnStateA</b> ( <b>\$newval</b> )
cpp	int <b>set_maxTimeOnStateA</b> ( s64 <b>newval</b> )
m	-(int) setMaxTimeOnStateA : (s64) <b>newval</b>
pas	function <b>set_maxTimeOnStateA</b> ( <b>newval</b> : int64): integer
vb	function <b>set_maxTimeOnStateA</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_maxTimeOnStateA</b> ( long <b>newval</b> )
java	int <b>set_maxTimeOnStateA</b> ( long <b>newval</b> )
py	def <b>set_maxTimeOnStateA</b> ( <b>newval</b> )
cmd	YRelay <b>target</b> <b>set_maxTimeOnStateA</b> <b>newval</b>

Use zero for no maximum time.

**Parameters :**

**newval** an integer

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**relay→set\_maxTimeOnStateB()****YRelay****relay→setMaxTimeOnStateB()****relay.set\_maxTimeOnStateB()**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

js	function <b>set_maxTimeOnStateB</b> ( <b>newval</b> )
nodejs	function <b>set_maxTimeOnStateB</b> ( <b>newval</b> )
php	function <b>set_maxTimeOnStateB</b> ( <b>\$newval</b> )
cpp	int <b>set_maxTimeOnStateB</b> ( s64 <b>newval</b> )
m	-(int) setMaxTimeOnStateB : (s64) <b>newval</b>
pas	function <b>set_maxTimeOnStateB</b> ( <b>newval</b> : int64): integer
vb	function <b>set_maxTimeOnStateB</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_maxTimeOnStateB</b> ( long <b>newval</b> )
java	int <b>set_maxTimeOnStateB</b> ( long <b>newval</b> )
py	def <b>set_maxTimeOnStateB</b> ( <b>newval</b> )
cmd	YRelay <b>target set_maxTimeOnStateB newval</b>

Use zero for no maximum time.

**Parameters :**

**newval** an integer

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**relay**→**set\_output()****YRelay****relay**→**setOutput()****relay.set\_output()**

Changes the output state of the relays, when used as a simple switch (single throw).

js	function <b>set_output</b> ( <b>newval</b> )
nodejs	function <b>set_output</b> ( <b>newval</b> )
php	function <b>set_output</b> ( <b>\$newval</b> )
cpp	int <b>set_output</b> ( Y_OUTPUT_enum <b>newval</b> )
m	-(int) setOutput : (Y_OUTPUT_enum) <b>newval</b>
pas	function <b>set_output</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_output</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_output</b> ( int <b>newval</b> )
java	int <b>set_output</b> ( int <b>newval</b> )
py	def <b>set_output</b> ( <b>newval</b> )
cmd	YRelay <b>target set_output newval</b>

**Parameters :**

**newval** either Y\_OUTPUT\_OFF or Y\_OUTPUT\_ON, according to the output state of the relays, when used as a simple switch (single throw)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**relay**→**set\_state()****YRelay****relay**→**setState()****relay.set\_state()**

Changes the state of the relays (A for the idle position, B for the active position).

js	function <b>set_state</b> ( <b>newval</b> )
nodejs	function <b>set_state</b> ( <b>newval</b> )
php	function <b>set_state</b> ( <b>\$newval</b> )
cpp	int <b>set_state</b> ( Y_STATE_enum <b>newval</b> )
m	-(int) <b>setState</b> : (Y_STATE_enum) <b>newval</b>
pas	function <b>set_state</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_state</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_state</b> ( int <b>newval</b> )
java	int <b>set_state</b> ( int <b>newval</b> )
py	def <b>set_state</b> ( <b>newval</b> )
cmd	YRelay <b>target set_state newval</b>

**Parameters :**

**newval** either Y\_STATE\_A or Y\_STATE\_B, according to the state of the relays (A for the idle position, B for the active position)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**relay→set\_stateAtPowerOn()****YRelay****relay→setStateAtPowerOn()****relay.set\_stateAtPowerOn()**

Preset the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no modification).

js	function <b>set_stateAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_stateAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_stateAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_stateAtPowerOn</b> ( Y_STATEATPOWERON_enum <b>newval</b> )
m	-(int) <b>setStateAtPowerOn</b> : (Y_STATEATPOWERON_enum) <b>newval</b>
pas	function <b>set_stateAtPowerOn</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_stateAtPowerOn</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_stateAtPowerOn</b> ( int <b>newval</b> )
java	int <b>set_stateAtPowerOn</b> ( int <b>newval</b> )
py	def <b>set_stateAtPowerOn</b> ( <b>newval</b> )
cmd	YRelay <b>target set_stateAtPowerOn newval</b>

Remember to call the matching module `saveToFlash( )` method, otherwise this call will have no effect.

**Parameters :**

**newval** a value among Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A and Y\_STATEATPOWERON\_B

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**relay**→**set\_userData()****YRelay****relay**→**setUserData()****relay.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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

**relay→wait\_async()****YRelay**

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.

## 3.36. Sensor function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_api.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;
php	require_once('yocto_api.php');
cpp	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
java	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *

### Global functions

#### yFindSensor(**func**)

Retrieves a sensor for a given identifier.

#### yFirstSensor()

Starts the enumeration of sensors currently accessible.

### YSensor methods

#### sensor→**calibrateFromPoints**(**rawValues**, **refValues**)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### sensor→**describe**()

Returns a short text that describes unambiguously the instance of the sensor in the form  
TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### sensor→**get\_advertisedValue**()

Returns the current value of the sensor (no more than 6 characters).

#### sensor→**get\_currentRawValue**()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### sensor→**get\_currentValue**()

Returns the current value of the measure.

#### sensor→**get\_errorMessage**()

Returns the error message of the latest error with the sensor.

#### sensor→**get\_errorType**()

Returns the numerical error code of the latest error with the sensor.

#### sensor→**get\_friendlyName**()

Returns a global identifier of the sensor in the format MODULE\_NAME . FUNCTION\_NAME.

#### sensor→**get\_functionDescriptor**()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### sensor→**get\_functionId**()

Returns the hardware identifier of the sensor, without reference to the module.

#### sensor→**get\_hardwareId**()

Returns the unique hardware identifier of the sensor in the form `SERIAL . FUNCTIONID`.

**sensor→get\_highestValue()**

Returns the maximal value observed for the measure since the device was started.

**sensor→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**sensor→get\_logicalName()**

Returns the logical name of the sensor.

**sensor→get\_lowestValue()**

Returns the minimal value observed for the measure since the device was started.

**sensor→get\_module()**

Gets the `YModule` object for the device on which the function is located.

**sensor→get\_module\_async(callback, context)**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

**sensor→get\_recordedData(startTime, endTime)**

Retrieves a `DataSet` object holding historical data for this sensor, for a specified time interval.

**sensor→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**sensor→get\_resolution()**

Returns the resolution of the measured values.

**sensor→get\_unit()**

Returns the measuring unit for the measure.

**sensor→get\_userData()**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

**sensor→isOnline()**

Checks if the sensor is currently reachable, without raising any error.

**sensor→isOnline\_async(callback, context)**

Checks if the sensor is currently reachable, without raising any error (asynchronous version).

**sensor→load(msValidity)**

Preloads the sensor cache with a specified validity duration.

**sensor→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

**sensor→load\_async(msValidity, callback, context)**

Preloads the sensor cache with a specified validity duration (asynchronous version).

**sensor→nextSensor()**

Continues the enumeration of sensors started using `yFirstSensor()`.

**sensor→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**sensor→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**sensor→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**sensor→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**sensor→set\_logicalName(newval)**

### 3. Reference

---

Changes the logical name of the sensor.

**sensor**→**set\_lowestValue**(**newval**)

Changes the recorded minimal value observed.

**sensor**→**set\_reportFrequency**(**newval**)

Changes the timed value notification frequency for this function.

**sensor**→**set\_resolution**(**newval**)

Changes the resolution of the measured physical values.

**sensor**→**set\_userData**(**data**)

Stores a user context provided as argument in the userData attribute of the function.

**sensor**→**wait\_async**(**callback**, **context**)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YSensor.FindSensor() yFindSensor()YSensor.FindSensor()

YSensor

Retrieves a sensor for a given identifier.

js	function <b>yFindSensor</b> ( <b>func</b> )
nodejs	function <b>FindSensor</b> ( <b>func</b> )
php	function <b>yFindSensor</b> ( <b>\$func</b> )
cpp	YSensor* <b>yFindSensor</b> ( string <b>func</b> )
m	+(YSensor*) <b>yFindSensor</b> : (NSString*) <b>func</b>
pas	function <b>yFindSensor</b> ( <b>func</b> : string): TYSensor
vb	function <b>yFindSensor</b> ( ByVal <b>func</b> As String) As YSensor
cs	YSensor <b>FindSensor</b> ( string <b>func</b> )
java	YSensor <b>FindSensor</b> ( String <b>func</b> )
py	def <b>FindSensor</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YSensor.isOnline()` to test if the sensor is indeed online at a given time. In case of ambiguity when looking for a sensor 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 sensor

### Returns :

a YSensor object allowing you to drive the sensor.

**YSensor.FirstSensor()****YSensor****yFirstSensor()YSensor.FirstSensor()**

Starts the enumeration of sensors currently accessible.

js	function <b>yFirstSensor</b> ( )
nodejs	function <b>FirstSensor</b> ( )
php	function <b>yFirstSensor</b> ( )
cpp	YSensor* <b>yFirstSensor</b> ( )
m	YSensor* <b>yFirstSensor</b> ( )
pas	function <b>yFirstSensor</b> ( ): TYSensor
vb	function <b>yFirstSensor</b> ( ) As YSensor
cs	YSensor <b>FirstSensor</b> ( )
java	YSensor <b>FirstSensor</b> ( )
py	def <b>FirstSensor</b> ( )

Use the method `YSensor.nextSensor( )` to iterate on next sensors.

**Returns :**

a pointer to a `YSensor` object, corresponding to the first sensor currently online, or a `null` pointer if there are none.



## sensor→calibrateFromPoints() sensor.calibrateFromPoints()

YSensor

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js      function calibrateFromPoints( rawValues, refValues)
node.js function calibrateFromPoints( rawValues, refValues)
php     function calibrateFromPoints( $rawValues, $refValues)
cpp     int calibrateFromPoints( vector<double> rawValues,
                                vector<double> refValues)

m       -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function calibrateFromPoints( rawValues: TDoubleArray,
                                refValues: TDoubleArray): LongInt

vb      procedure calibrateFromPoints( )

cs      int calibrateFromPoints( List<double> rawValues,
                                List<double> refValues)

java    int calibrateFromPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def calibrateFromPoints( rawValues, refValues)

cmd     YSensor target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**sensor→describe()****sensor.describe()****YSensor**

Returns a short text that describes unambiguously the instance of the sensor in the form  
 TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**sensor**→**get\_advertisedValue()****YSensor****sensor**→**advertisedValue()****sensor.get\_advertisedValue()**

Returns the current value of the sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YSensor <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**sensor**→**get\_currentRawValue()****YSensor****sensor**→**currentRawValue()****sensor.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YSensor <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**sensor**→**get\_currentValue()****YSensor****sensor**→**currentValue()****sensor.get\_currentValue()**

Returns the current value of the measure.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YSensor <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the measure

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**sensor**→**get\_errorMessage()****YSensor****sensor**→**errorMessage()****sensor.errorMessage()**

Returns the error message of the latest error with the sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the sensor object

**sensor**→**get\_errorType()****YSensor****sensor**→**errorType()****sensor.get\_errorType()**

Returns the numerical error code of the latest error with the sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the sensor object

**sensor**→**get\_friendlyName()****YSensor****sensor**→**friendlyName()****sensor.get\_friendlyName()**

Returns a global identifier of the sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the sensor (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.



**sensor**→**get\_functionDescriptor()****YSensor****sensor**→**functionDescriptor()****sensor.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**sensor**→**get\_functionId()****YSensor****sensor**→**functionId()****sensor.get\_functionId()**

Returns the hardware identifier of the sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**sensor**→**get\_hardwareId()****YSensor****sensor**→**hardwareId()****sensor.get\_hardwareId()**

Returns the unique hardware identifier of the sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**sensor**→**get\_highestValue()****YSensor****sensor**→**highestValue()****sensor.get\_highestValue()**

Returns the maximal value observed for the measure since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YSensor <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the measure since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**sensor**→**get\_logFrequency()****YSensor****sensor**→**logFrequency()****sensor.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YSensor <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**sensor**→**get\_logicalName()****YSensor****sensor**→**logicalName()****sensor.get\_logicalName()**

Returns the logical name of the sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YSensor <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**sensor**→**get\_lowestValue()****YSensor****sensor**→**lowestValue()****sensor.get\_lowestValue()**

Returns the minimal value observed for the measure since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YSensor <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the measure since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**sensor**→**get\_module()****YSensor****sensor**→**module()****sensor.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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



---

**sensor**→**get\_module\_async()****YSensor****sensor**→**module\_async()**

---

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**sensor**→**get\_recordedData()****YSensor****sensor**→**recordedData()****sensor.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YSensor <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**sensor→get\_reportFrequency()**  
**sensor→reportFrequency()**  
**sensor.get\_reportFrequency()**

**YSensor**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YSensor <b>target</b> <b>get_reportFrequency</b>

#### Returns :

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**sensor**→**get\_resolution()****YSensor****sensor**→**resolution()****sensor.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YSensor <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**sensor**→**get\_unit()****YSensor****sensor**→**unit()****sensor.get\_unit()**

Returns the measuring unit for the measure.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YSensor <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the measure

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**sensor**→**get\_userData()****YSensor****sensor**→**userData()****sensor.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**sensor**→**isOnline()****sensor.isOnline()****YSensor**

Checks if the sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the sensor 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 sensor.

**Returns :**

`true` if the sensor can be reached, and `false` otherwise

**sensor**→**isOnline\_async()****YSensor**

Checks if the sensor 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 sensor 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.



**sensor**→**load()****sensor.load()****YSensor**

Preloads the sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

## sensor→loadCalibrationPoints() sensor.loadCalibrationPoints()

YSensor

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YSensor target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**sensor→load\_async()****YSensor**

Preloads the sensor 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 traffic 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.

**sensor**→**nextSensor()****sensor.nextSensor()****YSensor**

Continues the enumeration of sensors started using `yFirstSensor()`.

js	function <b>nextSensor</b> ( )
nodejs	function <b>nextSensor</b> ( )
php	function <b>nextSensor</b> ( )
cpp	YSensor * <b>nextSensor</b> ( )
m	-(YSensor*) <b>nextSensor</b>
pas	function <b>nextSensor</b> ( ): TYSensor
vb	function <b>nextSensor</b> ( ) As YSensor
cs	YSensor <b>nextSensor</b> ( )
java	YSensor <b>nextSensor</b> ( )
py	def <b>nextSensor</b> ( )

**Returns :**

a pointer to a `YSensor` object, corresponding to a sensor currently online, or a `null` pointer if there are no more sensors to enumerate.

## sensor→registerTimedReportCallback() sensor.registerTimedReportCallback()

YSensor

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YSensorTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YSensorTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYSensorTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## sensor→registerValueCallback() sensor.registerValueCallback()

YSensor

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YSensorValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YSensorValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYSensorValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**sensor**→**set\_highestValue()****YSensor****sensor**→**setHighestValue()****sensor.set\_highestValue()**

Changes the recorded maximal value observed.

<code>js</code>	<code>function set_highestValue( newval)</code>
<code>nodejs</code>	<code>function set_highestValue( newval)</code>
<code>php</code>	<code>function set_highestValue( \$newval)</code>
<code>cpp</code>	<code>int set_highestValue( double newval)</code>
<code>m</code>	<code>-(int) setHighestValue : (double) newval</code>
<code>pas</code>	<code>function set_highestValue( newval: double): integer</code>
<code>vb</code>	<code>function set_highestValue( ByVal newval As Double) As Integer</code>
<code>cs</code>	<code>int set_highestValue( double newval)</code>
<code>java</code>	<code>int set_highestValue( double newval)</code>
<code>py</code>	<code>def set_highestValue( newval)</code>
<code>cmd</code>	<code>YSensor target set_highestValue newval</code>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**sensor**→**set\_logFrequency()****YSensor****sensor**→**setLogFrequency()****sensor.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

<code>js</code>	<code>function set_logFrequency( newval)</code>
<code>nodejs</code>	<code>function set_logFrequency( newval)</code>
<code>php</code>	<code>function set_logFrequency( \$newval)</code>
<code>cpp</code>	<code>int set_logFrequency( const string&amp; newval)</code>
<code>m</code>	<code>-(int) setLogFrequency : (NSString*) newval</code>
<code>pas</code>	<code>function set_logFrequency( newval: string): integer</code>
<code>vb</code>	<code>function set_logFrequency( ByVal newval As String) As Integer</code>
<code>cs</code>	<code>int set_logFrequency( string newval)</code>
<code>java</code>	<code>int set_logFrequency( String newval)</code>
<code>py</code>	<code>def set_logFrequency( newval)</code>
<code>cmd</code>	<code>YSensor target set_logFrequency newval</code>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**sensor**→**set\_logicalName()****YSensor****sensor**→**setLogicalName()****sensor.set\_logicalName()**

Changes the logical name of the sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YSensor <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**sensor**→**set\_lowestValue()****YSensor****sensor**→**setLowestValue()****sensor.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YSensor <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**sensor→set\_reportFrequency()**  
**sensor→setReportFrequency()**  
**sensor.set\_reportFrequency()**

**YSensor**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YSensor <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**sensor**→**set\_resolution()****YSensor****sensor**→**setResolution()****sensor.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YSensor <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**sensor**→**set\_userData()****YSensor****sensor**→**setUserData()****sensor.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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

**sensor**→**wait\_async()****YSensor**

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.

## 3.37. Servo function interface

Yoctopuce application programming interface allows you not only to move a servo to a given position, but also to specify the time interval in which the move should be performed. This makes it possible to synchronize two servos involved in a same move.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_servo.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YServo = yoctolib.YServo;
php	require_once('yocto_servo.php');
c++	#include "yocto_servo.h"
m	#import "yocto_servo.h"
pas	uses yocto_servo;
vb	yocto_servo.vb
cs	yocto_servo.cs
java	import com.yoctopuce.YoctoAPI.YServo;
py	from yocto_servo import *

### Global functions

#### yFindServo(func)

Retrieves a servo for a given identifier.

#### yFirstServo()

Starts the enumeration of servos currently accessible.

### YServo methods

#### servo→describe()

Returns a short text that describes unambiguously the instance of the servo in the form `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### servo→get\_advertisedValue()

Returns the current value of the servo (no more than 6 characters).

#### servo→get\_enabled()

Returns the state of the servos.

#### servo→get\_enabledAtPowerOn()

Returns the servo signal generator state at power up.

#### servo→get\_errorMessage()

Returns the error message of the latest error with the servo.

#### servo→get\_errorType()

Returns the numerical error code of the latest error with the servo.

#### servo→get\_friendlyName()

Returns a global identifier of the servo in the format `MODULE_NAME . FUNCTION_NAME`.

#### servo→get\_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

#### servo→get\_functionId()

Returns the hardware identifier of the servo, without reference to the module.

#### servo→get\_hardwareId()

Returns the unique hardware identifier of the servo in the form `SERIAL . FUNCTIONID`.

#### servo→get\_logicalName()

Returns the logical name of the servo.

**servo→get\_module()**

Gets the YModule object for the device on which the function is located.

**servo→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**servo→get\_neutral()**

Returns the duration in microseconds of a neutral pulse for the servo.

**servo→get\_position()**

Returns the current servo position.

**servo→get\_positionAtPowerOn()**

Returns the servo position at device power up.

**servo→get\_range()**

Returns the current range of use of the servo.

**servo→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**servo→isOnline()**

Checks if the servo is currently reachable, without raising any error.

**servo→isOnline\_async(callback, context)**

Checks if the servo is currently reachable, without raising any error (asynchronous version).

**servo→load(msValidity)**

Preloads the servo cache with a specified validity duration.

**servo→load\_async(msValidity, callback, context)**

Preloads the servo cache with a specified validity duration (asynchronous version).

**servo→move(target, ms\_duration)**

Performs a smooth move at constant speed toward a given position.

**servo→nextServo()**

Continues the enumeration of servos started using yFirstServo( ).

**servo→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**servo→set\_enabled(newval)**

Stops or starts the servo.

**servo→set\_enabledAtPowerOn(newval)**

Configure the servo signal generator state at power up.

**servo→set\_logicalName(newval)**

Changes the logical name of the servo.

**servo→set\_neutral(newval)**

Changes the duration of the pulse corresponding to the neutral position of the servo.

**servo→set\_position(newval)**

Changes immediately the servo driving position.

**servo→set\_positionAtPowerOn(newval)**

Configure the servo position at device power up.

**servo→set\_range(newval)**

Changes the range of use of the servo, specified in per cents.

**servo→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**servo→wait\_async(callback, context)**



Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YServo.FindServo() yFindServo()YServo.FindServo()

YServo

Retrieves a servo for a given identifier.

js	function <b>yFindServo</b> ( <b>func</b> )
nodejs	function <b>FindServo</b> ( <b>func</b> )
php	function <b>yFindServo</b> ( <b>\$func</b> )
cpp	YServo* <b>yFindServo</b> ( const string& <b>func</b> )
m	YServo* <b>yFindServo</b> ( NSString* <b>func</b> )
pas	function <b>yFindServo</b> ( <b>func</b> : string): TYServo
vb	function <b>yFindServo</b> ( ByVal <b>func</b> As String) As YServo
cs	YServo <b>FindServo</b> ( string <b>func</b> )
java	YServo <b>FindServo</b> ( String <b>func</b> )
py	def <b>FindServo</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the servo is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YServo.isOnline()` to test if the servo is indeed online at a given time. In case of ambiguity when looking for a servo 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 servo

### Returns :

a YServo object allowing you to drive the servo.

## YServo.FirstServo() yFirstServo()YServo.FirstServo()

## YServo

Starts the enumeration of servos currently accessible.

js	function <b>yFirstServo</b> ( )
nodejs	function <b>FirstServo</b> ( )
php	function <b>yFirstServo</b> ( )
cpp	YServo* <b>yFirstServo</b> ( )
m	YServo* <b>yFirstServo</b> ( )
pas	function <b>yFirstServo</b> ( ): TYServo
vb	function <b>yFirstServo</b> ( ) As YServo
cs	YServo <b>FirstServo</b> ( )
java	YServo <b>FirstServo</b> ( )
py	def <b>FirstServo</b> ( )

Use the method `YServo.nextServo( )` to iterate on next servos.

### Returns :

a pointer to a YServo object, corresponding to the first servo currently online, or a null pointer if there are none.

**servo→describe()****servo.describe()****YServo**

Returns a short text that describes unambiguously the instance of the servo in the form  
 TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 servo (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**servo**→**get\_advertisedValue()****YServo****servo**→**advertisedValue()****servo.get\_advertisedValue()**

Returns the current value of the servo (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YServo <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the servo (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**servo**→**get\_enabled()****servo**→**enabled()****servo.get\_enabled()**

Returns the state of the servos.

js	function <b>get_enabled</b> ( )
nodejs	function <b>get_enabled</b> ( )
php	function <b>get_enabled</b> ( )
cpp	Y_ENABLED_enum <b>get_enabled</b> ( )
m	-(Y_ENABLED_enum) enabled
pas	function <b>get_enabled</b> ( ): Integer
vb	function <b>get_enabled</b> ( ) As Integer
cs	int <b>get_enabled</b> ( )
java	int <b>get_enabled</b> ( )
py	def <b>get_enabled</b> ( )
cmd	YServo <b>target</b> <b>get_enabled</b>

**Returns :**

either Y\_ENABLED\_FALSE or Y\_ENABLED\_TRUE, according to the state of the servos

On failure, throws an exception or returns Y\_ENABLED\_INVALID.

**servo→get\_enabledAtPowerOn()****YServo****servo→enabledAtPowerOn()****servo.get\_enabledAtPowerOn()**

Returns the servo signal generator state at power up.

js	function <b>get_enabledAtPowerOn</b> ( )
nodejs	function <b>get_enabledAtPowerOn</b> ( )
php	function <b>get_enabledAtPowerOn</b> ( )
cpp	Y_ENABLEDATPOWERON_enum <b>get_enabledAtPowerOn</b> ( )
m	-(Y_ENABLEDATPOWERON_enum) enabledAtPowerOn
pas	function <b>get_enabledAtPowerOn</b> ( ): Integer
vb	function <b>get_enabledAtPowerOn</b> ( ) As Integer
cs	int <b>get_enabledAtPowerOn</b> ( )
java	int <b>get_enabledAtPowerOn</b> ( )
py	def <b>get_enabledAtPowerOn</b> ( )
cmd	YServo <b>target</b> <b>get_enabledAtPowerOn</b>

**Returns :**

either Y\_ENABLEDATPOWERON\_FALSE or Y\_ENABLEDATPOWERON\_TRUE, according to the servo signal generator state at power up

On failure, throws an exception or returns Y\_ENABLEDATPOWERON\_INVALID.

**servo**→**get\_errorMessage()****YServo****servo**→**errorMessage()****servo.get\_errorMessage()**

Returns the error message of the latest error with the servo.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the servo object



**servo**→**get\_errorType()****YServo****servo**→**errorType()****servo.get\_errorType()**

Returns the numerical error code of the latest error with the servo.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the servo object

**servo**→**get\_friendlyName()****YServo****servo**→**friendlyName()****servo.get\_friendlyName()**

Returns a global identifier of the servo in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the servo if they are defined, otherwise the serial number of the module and the hardware identifier of the servo (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the servo using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**servo→get\_functionDescriptor()**  
**servo→functionDescriptor()**  
**servo.get\_functionDescriptor()**

**YServo**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**servo**→**get\_functionId()****YServo****servo**→**functionId()****servo.get\_functionId()**

Returns the hardware identifier of the servo, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the servo (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**servo→get\_hardwareId()****YServo****servo→hardwareId()servo.get\_hardwareId()**

Returns the unique hardware identifier of the servo in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the servo. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the servo (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**servo**→**get\_logicalName()****YServo****servo**→**logicalName()****servo.get\_logicalName()**

Returns the logical name of the servo.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YServo <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the servo. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**servo→get\_module()****YServo****servo→module()****servo.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**servo→get\_module\_async()****YServo****servo→module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.



**servo**→**get\_neutral()****YServo****servo**→**neutral()****servo.get\_neutral()**

Returns the duration in microseconds of a neutral pulse for the servo.

js	function <b>get_neutral</b> ( )
nodejs	function <b>get_neutral</b> ( )
php	function <b>get_neutral</b> ( )
cpp	int <b>get_neutral</b> ( )
m	-(int) neutral
pas	function <b>get_neutral</b> ( ): LongInt
vb	function <b>get_neutral</b> ( ) As Integer
cs	int <b>get_neutral</b> ( )
java	int <b>get_neutral</b> ( )
py	def <b>get_neutral</b> ( )
cmd	YServo <b>target</b> <b>get_neutral</b>

**Returns :**

an integer corresponding to the duration in microseconds of a neutral pulse for the servo

On failure, throws an exception or returns Y\_NEUTRAL\_INVALID.

**servo**→**get\_position()****YServo****servo**→**position()****servo.get\_position()**

Returns the current servo position.

js	function <b>get_position</b> ( )
nodejs	function <b>get_position</b> ( )
php	function <b>get_position</b> ( )
cpp	int <b>get_position</b> ( )
m	-(int) position
pas	function <b>get_position</b> ( ): LongInt
vb	function <b>get_position</b> ( ) As Integer
cs	int <b>get_position</b> ( )
java	int <b>get_position</b> ( )
py	def <b>get_position</b> ( )
cmd	YServo <b>target</b> <b>get_position</b>

**Returns :**

an integer corresponding to the current servo position

On failure, throws an exception or returns Y\_POSITION\_INVALID.

**servo→get\_positionAtPowerOn()****YServo****servo→positionAtPowerOn()****servo.get\_positionAtPowerOn()**

Returns the servo position at device power up.

js	function <b>get_positionAtPowerOn</b> ( )
nodejs	function <b>get_positionAtPowerOn</b> ( )
php	function <b>get_positionAtPowerOn</b> ( )
cpp	int <b>get_positionAtPowerOn</b> ( )
m	-(int) positionAtPowerOn
pas	function <b>get_positionAtPowerOn</b> ( ): LongInt
vb	function <b>get_positionAtPowerOn</b> ( ) As Integer
cs	int <b>get_positionAtPowerOn</b> ( )
java	int <b>get_positionAtPowerOn</b> ( )
py	def <b>get_positionAtPowerOn</b> ( )
cmd	YServo <b>target</b> <b>get_positionAtPowerOn</b>

**Returns :**

an integer corresponding to the servo position at device power up

On failure, throws an exception or returns Y\_POSITIONATPOWERON\_INVALID.

**servo**→**get\_range()****servo**→**range()****servo.get\_range()**

Returns the current range of use of the servo.

js	function <b>get_range</b> ( )
nodejs	function <b>get_range</b> ( )
php	function <b>get_range</b> ( )
cpp	int <b>get_range</b> ( )
m	-(int) range
pas	function <b>get_range</b> ( ): LongInt
vb	function <b>get_range</b> ( ) As Integer
cs	int <b>get_range</b> ( )
java	int <b>get_range</b> ( )
py	def <b>get_range</b> ( )
cmd	YServo <b>target</b> <b>get_range</b>

**Returns :**

an integer corresponding to the current range of use of the servo

On failure, throws an exception or returns Y\_RANGE\_INVALID.

**servo**→**get\_userData()****YServo****servo**→**userData()****servo.userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**servo**→**isOnline()****servo.isOnline()****YServo**

Checks if the servo is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the servo 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 servo.

**Returns :**

`true` if the servo can be reached, and `false` otherwise

**servo→isOnline\_async()****YServo**

Checks if the servo 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 servo 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.

**servo→load()****servo.load()****YServo**

Preloads the servo cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.



**servo→load\_async()****YServo**

Preloads the servo 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 traffic 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.

**servo→move()****servo.move()****YServo**

Performs a smooth move at constant speed toward a given position.

js	function <b>move</b> ( <b>target</b> , <b>ms_duration</b> )
nodejs	function <b>move</b> ( <b>target</b> , <b>ms_duration</b> )
php	function <b>move</b> ( <b>\$target</b> , <b>\$ms_duration</b> )
cpp	int <b>move</b> ( int <b>target</b> , int <b>ms_duration</b> )
m	-(int) <b>move</b> : (int) <b>target</b> : (int) <b>ms_duration</b>
pas	function <b>move</b> ( <b>target</b> : LongInt, <b>ms_duration</b> : LongInt): integer
vb	function <b>move</b> ( ByVal <b>target</b> As Integer, ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>move</b> ( int <b>target</b> , int <b>ms_duration</b> )
java	int <b>move</b> ( int <b>target</b> , int <b>ms_duration</b> )
py	def <b>move</b> ( <b>target</b> , <b>ms_duration</b> )
cmd	YServo <b>target move target ms_duration</b>

**Parameters :**

**target** new position at the end of the move  
**ms\_duration** total duration of the move, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**servo→nextServo()****servo.nextServo()****YServo**

Continues the enumeration of servos started using `yFirstServo()`.

js	function <b>nextServo</b> ( )
nodejs	function <b>nextServo</b> ( )
php	function <b>nextServo</b> ( )
cpp	YServo * <b>nextServo</b> ( )
m	-(YServo*) <b>nextServo</b>
pas	function <b>nextServo</b> ( ): TYServo
vb	function <b>nextServo</b> ( ) As YServo
cs	YServo <b>nextServo</b> ( )
java	YServo <b>nextServo</b> ( )
py	def <b>nextServo</b> ( )

**Returns :**

a pointer to a `YServo` object, corresponding to a servo currently online, or a `null` pointer if there are no more servos to enumerate.

## servo→registerValueCallback() servo.registerValueCallback()

YServo

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YServoValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YServoValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYServoValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**servo**→**set\_enabled()****YServo****servo**→**setEnabled()****servo.set\_enabled()**

Stops or starts the servo.

js	function <b>set_enabled</b> ( <b>newval</b> )
nodejs	function <b>set_enabled</b> ( <b>newval</b> )
php	function <b>set_enabled</b> ( <b>\$newval</b> )
cpp	int <b>set_enabled</b> ( Y_ENABLED_enum <b>newval</b> )
m	-(int) <b>setEnabled</b> : (Y_ENABLED_enum) <b>newval</b>
pas	function <b>set_enabled</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_enabled</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_enabled</b> ( int <b>newval</b> )
java	int <b>set_enabled</b> ( int <b>newval</b> )
py	def <b>set_enabled</b> ( <b>newval</b> )
cmd	YServo <b>target set_enabled newval</b>

**Parameters :**

**newval** either Y\_ENABLED\_FALSE or Y\_ENABLED\_TRUE

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**servo→set\_enabledAtPowerOn()****YServo****servo→setEnabledAtPowerOn()****servo.set\_enabledAtPowerOn()**

Configure the servo signal generator state at power up.

js	function <b>set_enabledAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_enabledAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_enabledAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_enabledAtPowerOn</b> ( Y_ENABLEDATPOWERON_enum <b>newval</b> )
m	-(int) <b>setEnabledAtPowerOn</b> : (Y_ENABLEDATPOWERON_enum) <b>newval</b>
pas	function <b>set_enabledAtPowerOn</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_enabledAtPowerOn</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_enabledAtPowerOn</b> ( int <b>newval</b> )
java	int <b>set_enabledAtPowerOn</b> ( int <b>newval</b> )
py	def <b>set_enabledAtPowerOn</b> ( <b>newval</b> )
cmd	YServo <b>target</b> <b>set_enabledAtPowerOn</b> <b>newval</b>

Remember to call the matching module `saveToFlash()` method, otherwise this call will have no effect.

**Parameters :**

**newval** either Y\_ENABLEDATPOWERON\_FALSE or Y\_ENABLEDATPOWERON\_TRUE

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**servo→set\_logicalName()****YServo****servo→setLogicalName()****servo.set\_logicalName()**

Changes the logical name of the servo.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YServo <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 servo.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**servo**→**set\_neutral()****servo**→**setNeutral()****servo.set\_neutral()**

Changes the duration of the pulse corresponding to the neutral position of the servo.

js	function <b>set_neutral</b> ( <b>newval</b> )
nodejs	function <b>set_neutral</b> ( <b>newval</b> )
php	function <b>set_neutral</b> ( <b>\$newval</b> )
cpp	int <b>set_neutral</b> ( int <b>newval</b> )
m	-(int) setNeutral : (int) <b>newval</b>
pas	function <b>set_neutral</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_neutral</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_neutral</b> ( int <b>newval</b> )
java	int <b>set_neutral</b> ( int <b>newval</b> )
py	def <b>set_neutral</b> ( <b>newval</b> )
cmd	YServo <b>target set_neutral newval</b>

The duration is specified in microseconds, and the standard value is 1500 [us]. This setting makes it possible to shift the range of use of the servo. Be aware that using a range higher than what is supported by the servo is likely to damage the servo.

**Parameters :**

**newval** an integer corresponding to the duration of the pulse corresponding to the neutral position of the servo

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**servo→set\_position()****YServo****servo→setPosition()****servo.set\_position()**

Changes immediately the servo driving position.

js	function <b>set_position</b> ( <b>newval</b> )
nodejs	function <b>set_position</b> ( <b>newval</b> )
php	function <b>set_position</b> ( <b>\$newval</b> )
cpp	int <b>set_position</b> ( int <b>newval</b> )
m	-(int) setPosition : (int) <b>newval</b>
pas	function <b>set_position</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_position</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_position</b> ( int <b>newval</b> )
java	int <b>set_position</b> ( int <b>newval</b> )
py	def <b>set_position</b> ( <b>newval</b> )
cmd	YServo <b>target set_position newval</b>

**Parameters :**

**newval** an integer corresponding to immediately the servo driving position

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**servo**→**set\_positionAtPowerOn()****YServo****servo**→**setPositionAtPowerOn()****servo.set\_positionAtPowerOn()**

Configure the servo position at device power up.

js	function <b>set_positionAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_positionAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_positionAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_positionAtPowerOn</b> ( int <b>newval</b> )
m	-(int) setPositionAtPowerOn : (int) <b>newval</b>
pas	function <b>set_positionAtPowerOn</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_positionAtPowerOn</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_positionAtPowerOn</b> ( int <b>newval</b> )
java	int <b>set_positionAtPowerOn</b> ( int <b>newval</b> )
py	def <b>set_positionAtPowerOn</b> ( <b>newval</b> )
cmd	YServo <b>target</b> <b>set_positionAtPowerOn</b> <b>newval</b>

Remember to call the matching module `saveToFlash()` method, otherwise this call will have no effect.

**Parameters :**

**newval** an integer

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**servo→set\_range()****YServo****servo→setRange()****servo.set\_range()**

Changes the range of use of the servo, specified in per cents.

js	function <b>set_range</b> ( <b>newval</b> )
nodejs	function <b>set_range</b> ( <b>newval</b> )
php	function <b>set_range</b> ( <b>\$newval</b> )
cpp	int <b>set_range</b> ( int <b>newval</b> )
m	-(int) setRange : (int) <b>newval</b>
pas	function <b>set_range</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_range</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_range</b> ( int <b>newval</b> )
java	int <b>set_range</b> ( int <b>newval</b> )
py	def <b>set_range</b> ( <b>newval</b> )
cmd	YServo <b>target set_range newval</b>

A range of 100% corresponds to a standard control signal, that varies from 1 [ms] to 2 [ms], When using a servo that supports a double range, from 0.5 [ms] to 2.5 [ms], you can select a range of 200%. Be aware that using a range higher than what is supported by the servo is likely to damage the servo.

**Parameters :**

**newval** an integer corresponding to the range of use of the servo, specified in per cents

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**servo**→**set\_userData()****servo**→**setUserData()****servo.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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

**servo→wait\_async()****YServo**

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.

## 3.38. Temperature function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_temperature.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YTemperature = yoctolib.YTemperature;
php	require_once('yocto_temperature.php');
c++	#include "yocto_temperature.h"
m	#import "yocto_temperature.h"
pas	uses yocto_temperature;
vb	yocto_temperature.vb
cs	yocto_temperature.cs
java	import com.yoctopuce.YoctoAPI.YTemperature;
py	from yocto_temperature import *

### Global functions

#### yFindTemperature(func)

Retrieves a temperature sensor for a given identifier.

#### yFirstTemperature()

Starts the enumeration of temperature sensors currently accessible.

### YTemperature methods

#### temperature→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### temperature→describe()

Returns a short text that describes unambiguously the instance of the temperature sensor in the form `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### temperature→get\_advertisedValue()

Returns the current value of the temperature sensor (no more than 6 characters).

#### temperature→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### temperature→get\_currentValue()

Returns the current value of the temperature.

#### temperature→get\_errorMessage()

Returns the error message of the latest error with the temperature sensor.

#### temperature→get\_errorType()

Returns the numerical error code of the latest error with the temperature sensor.

#### temperature→get\_friendlyName()

Returns a global identifier of the temperature sensor in the format `MODULE_NAME . FUNCTION_NAME`.

#### temperature→get\_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

#### temperature→get\_functionId()

Returns the hardware identifier of the temperature sensor, without reference to the module.

#### temperature→get\_hardwareId()

Returns the unique hardware identifier of the temperature sensor in the form `SERIAL . FUNCTIONID`.

**temperature→get\_highestValue()**

Returns the maximal value observed for the temperature since the device was started.

**temperature→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**temperature→get\_logicalName()**

Returns the logical name of the temperature sensor.

**temperature→get\_lowestValue()**

Returns the minimal value observed for the temperature since the device was started.

**temperature→get\_module()**

Gets the YModule object for the device on which the function is located.

**temperature→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**temperature→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**temperature→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**temperature→get\_resolution()**

Returns the resolution of the measured values.

**temperature→get\_sensorType()**

Returns the temperature sensor type.

**temperature→get\_unit()**

Returns the measuring unit for the temperature.

**temperature→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**temperature→isOnline()**

Checks if the temperature sensor is currently reachable, without raising any error.

**temperature→isOnline\_async(callback, context)**

Checks if the temperature sensor is currently reachable, without raising any error (asynchronous version).

**temperature→load(msValidity)**

Preloads the temperature sensor cache with a specified validity duration.

**temperature→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**temperature→load\_async(msValidity, callback, context)**

Preloads the temperature sensor cache with a specified validity duration (asynchronous version).

**temperature→nextTemperature()**

Continues the enumeration of temperature sensors started using yFirstTemperature().

**temperature→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**temperature→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**temperature→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**temperature→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

### 3. Reference

---

**temperature→set\_logicalName(newval)**

Changes the logical name of the temperature sensor.

**temperature→set\_lowestValue(newval)**

Changes the recorded minimal value observed.

**temperature→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**temperature→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**temperature→set\_sensorType(newval)**

Modify the temperature sensor type.

**temperature→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**temperature→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.



## YTemperature.FindTemperature() yFindTemperature()YTemperature.FindTemperature()

## YTemperature

Retrieves a temperature sensor for a given identifier.

js	function <b>yFindTemperature</b> ( <b>func</b> )
nodejs	function <b>FindTemperature</b> ( <b>func</b> )
php	function <b>yFindTemperature</b> ( <b>\$func</b> )
cpp	YTemperature* <b>yFindTemperature</b> ( const string& <b>func</b> )
m	YTemperature* <b>yFindTemperature</b> ( NSString* <b>func</b> )
pas	function <b>yFindTemperature</b> ( <b>func</b> : string): TYTemperature
vb	function <b>yFindTemperature</b> ( ByVal <b>func</b> As String) As YTemperature
cs	YTemperature <b>FindTemperature</b> ( string <b>func</b> )
java	YTemperature <b>FindTemperature</b> ( String <b>func</b> )
py	def <b>FindTemperature</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the temperature sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YTemperature.IsOnline()` to test if the temperature sensor is indeed online at a given time. In case of ambiguity when looking for a temperature sensor 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 temperature sensor

### Returns :

a `YTemperature` object allowing you to drive the temperature sensor.

**YTemperature.FirstTemperature()****YTemperature****yFirstTemperature()YTemperature.FirstTemperature()**

Starts the enumeration of temperature sensors currently accessible.

js	function <b>yFirstTemperature</b> ( )
nodejs	function <b>FirstTemperature</b> ( )
php	function <b>yFirstTemperature</b> ( )
cpp	YTemperature* <b>yFirstTemperature</b> ( )
m	YTemperature* <b>yFirstTemperature</b> ( )
pas	function <b>yFirstTemperature</b> ( ): TYTemperature
vb	function <b>yFirstTemperature</b> ( ) As YTemperature
cs	YTemperature <b>FirstTemperature</b> ( )
java	YTemperature <b>FirstTemperature</b> ( )
py	def <b>FirstTemperature</b> ( )

Use the method `YTemperature.nextTemperature( )` to iterate on next temperature sensors.

**Returns :**

a pointer to a `YTemperature` object, corresponding to the first temperature sensor currently online, or a `null` pointer if there are none.

## temperature→calibrateFromPoints() temperature.calibrateFromPoints()

YTemperature

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js      function calibrateFromPoints( rawValues, refValues)
node.js function calibrateFromPoints( rawValues, refValues)
php     function calibrateFromPoints( $rawValues, $refValues)
cpp     int calibrateFromPoints( vector<double> rawValues,
                                vector<double> refValues)

m       -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function calibrateFromPoints( rawValues: TDoubleArray,
                                refValues: TDoubleArray): LongInt

vb      procedure calibrateFromPoints( )

cs      int calibrateFromPoints( List<double> rawValues,
                                List<double> refValues)

java    int calibrateFromPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def calibrateFromPoints( rawValues, refValues)

cmd     YTemperature target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**temperature→describe()temperature.describe()****YTemperature**

Returns a short text that describes unambiguously the instance of the temperature sensor in the form `TYPE (NAME) =SERIAL.FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 temperature sensor (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**temperature**→**get\_advertisedValue()****YTemperature****temperature**→**advertisedValue()****temperature.get\_advertisedValue()**

Returns the current value of the temperature sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YTemperature <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the temperature sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**temperature**→**get\_currentRawValue()****YTemperature****temperature**→**currentRawValue()****temperature.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YTemperature <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**temperature**→**get\_currentValue()****YTemperature****temperature**→**currentValue()****temperature.get\_currentValue()**

Returns the current value of the temperature.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YTemperature <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the temperature

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**temperature**→**get\_errorMessage()****YTemperature****temperature**→**errorMessage()****temperature.get\_errorMessage()**

Returns the error message of the latest error with the temperature sensor.

<code>js</code>	<code>function get_errorMessage( )</code>
<code>nodejs</code>	<code>function get_errorMessage( )</code>
<code>php</code>	<code>function get_errorMessage( )</code>
<code>cpp</code>	<code>string get_errorMessage( )</code>
<code>m</code>	<code>-(NSString*) errorMessage</code>
<code>pas</code>	<code>function get_errorMessage( ): string</code>
<code>vb</code>	<code>function get_errorMessage( ) As String</code>
<code>cs</code>	<code>string get_errorMessage( )</code>
<code>java</code>	<code>String get_errorMessage( )</code>
<code>py</code>	<code>def get_errorMessage( )</code>

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the temperature sensor object



**temperature**→**get\_errorType()****YTemperature****temperature**→**errorType()****temperature.get\_errorType()**

Returns the numerical error code of the latest error with the temperature sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the temperature sensor object

**temperature→get\_friendlyName()****YTemperature****temperature→friendlyName()****temperature.get\_friendlyName()**

Returns a global identifier of the temperature sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the temperature sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the temperature sensor (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the temperature sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**temperature**→**get\_functionDescriptor()****YTemperature****temperature**→**functionDescriptor()****temperature.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**temperature**→**get\_functionId()****YTemperature****temperature**→**functionId()****temperature.get\_functionId()**

Returns the hardware identifier of the temperature sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the temperature sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**temperature**→**get\_hardwareId()****YTemperature****temperature**→**hardwareId()****temperature.get\_hardwareId()**

Returns the unique hardware identifier of the temperature sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the temperature sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the temperature sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**temperature**→**get\_highestValue()****YTemperature****temperature**→**highestValue()****temperature.get\_highestValue()**

Returns the maximal value observed for the temperature since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YTemperature <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the temperature since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**temperature**→**get\_logFrequency()****YTemperature****temperature**→**logFrequency()****temperature.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
c++	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YTemperature <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**temperature→get\_logicalName()****YTemperature****temperature→logicalName()****temperature.get\_logicalName()**

Returns the logical name of the temperature sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YTemperature <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the temperature sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**temperature**→**get\_lowestValue()****YTemperature****temperature**→**lowestValue()****temperature.get\_lowestValue()**

Returns the minimal value observed for the temperature since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YTemperature <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the temperature since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**temperature**→**get\_module()****YTemperature****temperature**→**module()****temperature.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

---

**temperature**→**get\_module\_async()****YTemperature****temperature**→**module\_async()**

---

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**temperature→get\_recordedData()****YTemperature****temperature→recordedData()****temperature.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
c++	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YTemperature <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**temperature→get\_reportFrequency()****YTemperature****temperature→reportFrequency()****temperature.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YTemperature <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**temperature**→**get\_resolution()****YTemperature****temperature**→**resolution()****temperature.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YTemperature <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**temperature**→**get\_sensorType()****YTemperature****temperature**→**sensorType()****temperature.get\_sensorType()**

Returns the temperature sensor type.

js	function <b>get_sensorType</b> ( )
nodejs	function <b>get_sensorType</b> ( )
php	function <b>get_sensorType</b> ( )
cpp	Y_SENSORTYPE_enum <b>get_sensorType</b> ( )
m	-(Y_SENSORTYPE_enum) sensorType
pas	function <b>get_sensorType</b> ( ): Integer
vb	function <b>get_sensorType</b> ( ) As Integer
cs	int <b>get_sensorType</b> ( )
java	int <b>get_sensorType</b> ( )
py	def <b>get_sensorType</b> ( )
cmd	YTemperature <b>target</b> <b>get_sensorType</b>

**Returns :**

a value among Y\_SENSORTYPE\_DIGITAL, Y\_SENSORTYPE\_TYPE\_K, Y\_SENSORTYPE\_TYPE\_E, Y\_SENSORTYPE\_TYPE\_J, Y\_SENSORTYPE\_TYPE\_N, Y\_SENSORTYPE\_TYPE\_R, Y\_SENSORTYPE\_TYPE\_S, Y\_SENSORTYPE\_TYPE\_T, Y\_SENSORTYPE\_PT100\_4WIRES, Y\_SENSORTYPE\_PT100\_3WIRES and Y\_SENSORTYPE\_PT100\_2WIRES corresponding to the temperature sensor type

On failure, throws an exception or returns Y\_SENSORTYPE\_INVALID.

**temperature**→**get\_unit()****YTemperature****temperature**→**unit()****temperature.get\_unit()**

Returns the measuring unit for the temperature.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YTemperature <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the temperature

On failure, throws an exception or returns Y\_UNIT\_INVALID.



**temperature**→**get\_userdata()****YTemperature****temperature**→**userData()****temperature.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**temperature**→**isOnline()****temperature.isOnline()****YTemperature**

Checks if the temperature sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the temperature sensor 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 temperature sensor.

**Returns :**

`true` if the temperature sensor can be reached, and `false` otherwise

**temperature→isOnline\_async()****YTemperature**

Checks if the temperature sensor 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 temperature sensor 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.

**temperature→load()temperature.load()****YTemperature**

Preloads the temperature sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

## temperature→loadCalibrationPoints() temperature.loadCalibrationPoints()

YTemperature

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js      function loadCalibrationPoints( rawValues, refValues)
nodejs  function loadCalibrationPoints( rawValues, refValues)
php     function loadCalibrationPoints( &$rawValues, &$refValues)
cpp     int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m       -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )
cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)
cmd     YTemperature target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**temperature→load\_async()****YTemperature**

Preloads the temperature sensor 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 traffic 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.

**temperature**→**nextTemperature()**  
**temperature.nextTemperature()**

**YTemperature**

Continues the enumeration of temperature sensors started using `yFirstTemperature()`.

js	function <b>nextTemperature</b> ( )
nodejs	function <b>nextTemperature</b> ( )
php	function <b>nextTemperature</b> ( )
cpp	YTemperature * <b>nextTemperature</b> ( )
m	-(YTemperature*) <b>nextTemperature</b>
pas	function <b>nextTemperature</b> ( ): TYTemperature
vb	function <b>nextTemperature</b> ( ) As YTemperature
cs	YTemperature <b>nextTemperature</b> ( )
java	YTemperature <b>nextTemperature</b> ( )
py	def <b>nextTemperature</b> ( )

#### Returns :

a pointer to a YTemperature object, corresponding to a temperature sensor currently online, or a null pointer if there are no more temperature sensors to enumerate.

## temperature→registerTimedReportCallback() temperature.registerTimedReportCallback()

YTemperature

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YTemperatureTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YTemperatureTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYTemperatureTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.



## temperature→registerValueCallback() temperature.registerValueCallback()

## YTemperature

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YTemperatureValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YTemperatureValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYTemperatureValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**temperature**→**set\_highestValue()****YTemperature****temperature**→**setHighestValue()****temperature.set\_highestValue()**

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YTemperature <b>target</b> <b>set_highestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**temperature→set\_logFrequency()**  
**temperature→setLogFrequency()**  
**temperature.set\_logFrequency()**

**YTemperature**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YTemperature <b>target</b> <b>set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the datalogger recording frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**temperature→set\_logicalName()****YTemperature****temperature→setLogicalName()****temperature.set\_logicalName()**

Changes the logical name of the temperature sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YTemperature <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 temperature sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**temperature**→**set\_lowestValue()****YTemperature****temperature**→**setLowestValue()****temperature.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YTemperature <b>target</b> <b>set_lowestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

temperature→set\_reportFrequency()

YTemperature

temperature→setReportFrequency()

temperature.set\_reportFrequency()

Changes the timed value notification frequency for this function.

js	function set_reportFrequency( newval)
nodejs	function set_reportFrequency( newval)
php	function set_reportFrequency( \$newval)
cpp	int set_reportFrequency( const string& newval)
m	-(int) setReportFrequency : (NSString*) newval
pas	function set_reportFrequency( newval: string): integer
vb	function set_reportFrequency( ByVal newval As String) As Integer
cs	int set_reportFrequency( string newval)
java	int set_reportFrequency( String newval)
py	def set_reportFrequency( newval)
cmd	YTemperature target set_reportFrequency newval

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

#### Parameters :

**newval** a string corresponding to the timed value notification frequency for this function

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**temperature→set\_resolution()****YTemperature****temperature→setResolution()****temperature.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YTemperature <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**temperature→set\_sensorType()****YTemperature****temperature→setSensorType()****temperature.set\_sensorType()**

Modify the temperature sensor type.

js	function <b>set_sensorType</b> ( <b>newval</b> )
nodejs	function <b>set_sensorType</b> ( <b>newval</b> )
php	function <b>set_sensorType</b> ( <b>\$newval</b> )
cpp	int <b>set_sensorType</b> ( Y_SENSORTYPE_enum <b>newval</b> )
m	-(int) setSensorType : (Y_SENSORTYPE_enum) <b>newval</b>
pas	function <b>set_sensorType</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_sensorType</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_sensorType</b> ( int <b>newval</b> )
java	int <b>set_sensorType</b> ( int <b>newval</b> )
py	def <b>set_sensorType</b> ( <b>newval</b> )
cmd	YTemperature <b>target</b> <b>set_sensorType</b> <b>newval</b>

This function is used to to define the type of thermocouple (K,E...) used with the device. This will have no effect if module is using a digital sensor. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

**Parameters :**

**newval** a value among Y\_SENSORTYPE\_DIGITAL, Y\_SENSORTYPE\_TYPE\_K, Y\_SENSORTYPE\_TYPE\_E, Y\_SENSORTYPE\_TYPE\_J, Y\_SENSORTYPE\_TYPE\_N, Y\_SENSORTYPE\_TYPE\_R, Y\_SENSORTYPE\_TYPE\_S, Y\_SENSORTYPE\_TYPE\_T, Y\_SENSORTYPE\_PT100\_4WIRES, Y\_SENSORTYPE\_PT100\_3WIRES and Y\_SENSORTYPE\_PT100\_2WIRES

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**temperature→set\_userdata()****YTemperature****temperature→setUserData()****temperature.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**temperature→wait\_async()****YTemperature**

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.

## 3.39. Tilt function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<code>&lt;script type='text/javascript' src='yocto_tilt.js'&gt;&lt;/script&gt;</code>
nodejs	<code>var yoctolib = require('yoctolib'); var YTilt = yoctolib.YTilt;</code>
php	<code>require_once('yocto_tilt.php');</code>
c++	<code>#include "yocto_tilt.h"</code>
m	<code>#import "yocto_tilt.h"</code>
pas	<code>uses yocto_tilt;</code>
vb	<code>yocto_tilt.vb</code>
cs	<code>yocto_tilt.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YTilt;</code>
py	<code>from yocto_tilt import *</code>

### Global functions

#### yFindTilt(func)

Retrieves a tilt sensor for a given identifier.

#### yFirstTilt()

Starts the enumeration of tilt sensors currently accessible.

### YTilt methods

#### tilt→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### tilt→describe()

Returns a short text that describes unambiguously the instance of the tilt sensor in the form `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### tilt→get\_advertisedValue()

Returns the current value of the tilt sensor (no more than 6 characters).

#### tilt→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### tilt→get\_currentValue()

Returns the current value of the inclination.

#### tilt→get\_errorMessage()

Returns the error message of the latest error with the tilt sensor.

#### tilt→get\_errorType()

Returns the numerical error code of the latest error with the tilt sensor.

#### tilt→get\_friendlyName()

Returns a global identifier of the tilt sensor in the format `MODULE_NAME . FUNCTION_NAME`.

#### tilt→get\_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

#### tilt→get\_functionId()

Returns the hardware identifier of the tilt sensor, without reference to the module.

#### tilt→get\_hardwareId()

Returns the unique hardware identifier of the tilt sensor in the form `SERIAL . FUNCTIONID`.

**tilt→get\_highestValue()**

Returns the maximal value observed for the inclination since the device was started.

**tilt→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**tilt→get\_logicalName()**

Returns the logical name of the tilt sensor.

**tilt→get\_lowestValue()**

Returns the minimal value observed for the inclination since the device was started.

**tilt→get\_module()**

Gets the YModule object for the device on which the function is located.

**tilt→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**tilt→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**tilt→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**tilt→get\_resolution()**

Returns the resolution of the measured values.

**tilt→get\_unit()**

Returns the measuring unit for the inclination.

**tilt→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**tilt→isOnline()**

Checks if the tilt sensor is currently reachable, without raising any error.

**tilt→isOnline\_async(callback, context)**

Checks if the tilt sensor is currently reachable, without raising any error (asynchronous version).

**tilt→load(msValidity)**

Preloads the tilt sensor cache with a specified validity duration.

**tilt→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**tilt→load\_async(msValidity, callback, context)**

Preloads the tilt sensor cache with a specified validity duration (asynchronous version).

**tilt→nextTilt()**

Continues the enumeration of tilt sensors started using yFirstTilt().

**tilt→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**tilt→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**tilt→set\_highestValue(newval)**

Changes the recorded maximal value observed.

**tilt→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**tilt→set\_logicalName(newval)**

Changes the logical name of the tilt sensor.

**tilt→set\_lowestValue(newval)**

Changes the recorded minimal value observed.

**tilt→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**tilt→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**tilt→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**tilt→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YTilt.FindTilt()****YTilt****yFindTilt()YTilt.FindTilt()**

Retrieves a tilt sensor for a given identifier.

js	function <b>yFindTilt</b> ( <b>func</b> )
nodejs	function <b>FindTilt</b> ( <b>func</b> )
php	function <b>yFindTilt</b> ( <b>\$func</b> )
cpp	YTilt* <b>yFindTilt</b> ( const string& <b>func</b> )
m	YTilt* <b>yFindTilt</b> ( NSString* <b>func</b> )
pas	function <b>yFindTilt</b> ( <b>func</b> : string): TYTilt
vb	function <b>yFindTilt</b> ( ByVal <b>func</b> As String) As YTilt
cs	YTilt <b>FindTilt</b> ( string <b>func</b> )
java	YTilt <b>FindTilt</b> ( String <b>func</b> )
py	def <b>FindTilt</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the tilt sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YTilt.isOnline()` to test if the tilt sensor is indeed online at a given time. In case of ambiguity when looking for a tilt sensor 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 tilt sensor

**Returns :**

a `YTilt` object allowing you to drive the tilt sensor.

**YTilt.FirstTilt()****YTilt****yFirstTilt()****YTilt.FirstTilt()**

Starts the enumeration of tilt sensors currently accessible.

js	function <b>yFirstTilt</b> ( )
nodejs	function <b>FirstTilt</b> ( )
php	function <b>yFirstTilt</b> ( )
cpp	<b>YTilt*</b> <b>yFirstTilt</b> ( )
m	<b>YTilt*</b> <b>yFirstTilt</b> ( )
pas	function <b>yFirstTilt</b> ( ): <b>TYTilt</b>
vb	function <b>yFirstTilt</b> ( ) As <b>YTilt</b>
cs	<b>YTilt</b> <b>FirstTilt</b> ( )
java	<b>YTilt</b> <b>FirstTilt</b> ( )
py	def <b>FirstTilt</b> ( )

Use the method `YTilt.nextTilt()` to iterate on next tilt sensors.

**Returns :**

a pointer to a `YTilt` object, corresponding to the first tilt sensor currently online, or a `null` pointer if there are none.

**tilt→calibrateFromPoints()****tilt.calibrateFromPoints()****YTilt**

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)
m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
    : (NSMutableArray*) refValues
pas function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)
py def calibrateFromPoints( rawValues, refValues)
cmd YTilt target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Parameters :**

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**tilt→describe()****tilt.describe()****YTilt**

Returns a short text that describes unambiguously the instance of the tilt sensor in the form `TYPE (NAME) = SERIAL.FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

More precisely, `TYPE` is the type of the function, `NAME` is 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(BadCustomName.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 tilt sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**tilt**→**get\_advertisedValue()****YTilt****tilt**→**advertisedValue()****tilt.get\_advertisedValue()**

Returns the current value of the tilt sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YTilt <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the tilt sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**tilt**→**get\_currentRawValue()****YTilt****tilt**→**currentRawValue()****tilt.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YTilt <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**tilt**→**get\_currentValue()****tilt**→**currentValue()****tilt.get\_currentValue()**

Returns the current value of the inclination.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YTilt <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current value of the inclination

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**tilt**→**get\_errorMessage()****YTilt****tilt**→**errorMessage()****tilt.get\_errorMessage()**

Returns the error message of the latest error with the tilt sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the tilt sensor object

**tilt**→**get\_errorType()****YTilt****tilt**→**errorType()****tilt.get\_errorType()**

Returns the numerical error code of the latest error with the tilt sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the tilt sensor object

**tilt→get\_friendlyName()****YTilt****tilt→friendlyName()tilt.get\_friendlyName()**

Returns a global identifier of the tilt sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the tilt sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the tilt sensor (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the tilt sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

tilt→get\_functionDescriptor()

YTilt

tilt→functionDescriptor()tilt.get\_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.



**tilt→get\_functionId()****YTilt****tilt→functionId()tilt.get\_functionId()**

Returns the hardware identifier of the tilt sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) functionId
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the tilt sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**tilt**→**get\_hardwareId()****tilt**→**hardwareId()****tilt.get\_hardwareId()**

Returns the unique hardware identifier of the tilt sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
c++	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the tilt sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the tilt sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**tilt**→**get\_highestValue()****YTilt****tilt**→**highestValue()****tilt.get\_highestValue()**

Returns the maximal value observed for the inclination since the device was started.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YTilt <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the inclination since the device was started

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**tilt→get\_logFrequency()****YTilt****tilt→logFrequency()****tilt.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YTilt <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**tilt→get\_logicalName()****YTilt****tilt→logicalName()tilt.get\_logicalName()**

Returns the logical name of the tilt sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YTilt <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the tilt sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**tilt**→**get\_lowestValue()****tilt**→**lowestValue()****tilt.get\_lowestValue()**

Returns the minimal value observed for the inclination since the device was started.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YTilt <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the inclination since the device was started

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**tilt→get\_module()****YTilt****tilt→module()tilt.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**tilt→get\_module\_async()****tilt→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.



**tilt→get\_recordedData()****YTilt****tilt→recordedData()tilt.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YTilt <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

#### Parameters :

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

#### Returns :

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**tilt→get\_reportFrequency()****YTilt****tilt→reportFrequency()****tilt.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YTilt <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**tilt**→**get\_resolution()****YTilt****tilt**→**resolution()****tilt.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YTilt <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**tilt**→**get\_unit()****tilt**→**unit()****tilt.get\_unit()**

Returns the measuring unit for the inclination.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YTilt <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the inclination

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**tilt**→**get\_userData()****YTilt****tilt**→**userData()****tilt.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

Checks if the tilt sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the tilt sensor 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 tilt sensor.

**Returns :**

true if the tilt sensor can be reached, and false otherwise

**tilt→isOnline\_async()****YTilt**

Checks if the tilt sensor 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 tilt sensor 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.

**tilt→load()tilt.load()****YTilt**

Preloads the tilt sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.



## tilt→loadCalibrationPoints() tilt.loadCalibrationPoints()

YTilt

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js      function loadCalibrationPoints( rawValues, refValues)
nodejs  function loadCalibrationPoints( rawValues, refValues)
php     function loadCalibrationPoints( &$rawValues, &$refValues)
cpp     int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m       -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues

pas     function loadCalibrationPoints( var rawValues: TDoubleArray,
                                var refValues: TDoubleArray): LongInt

vb      procedure loadCalibrationPoints( )
cs      int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)

java    int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)

py      def loadCalibrationPoints( rawValues, refValues)
cmd     YTilt target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**tilt→load\_async()**

Preloads the tilt sensor 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 traffic 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.

**tilt→nextTilt()tilt.nextTilt()****YTilt**

Continues the enumeration of tilt sensors started using `yFirstTilt()`.

js	function <b>nextTilt</b> ( )
nodejs	function <b>nextTilt</b> ( )
php	function <b>nextTilt</b> ( )
cpp	YTilt * <b>nextTilt</b> ( )
m	-(YTilt*) <b>nextTilt</b>
pas	function <b>nextTilt</b> ( ): TYTilt
vb	function <b>nextTilt</b> ( ) As YTilt
cs	YTilt <b>nextTilt</b> ( )
java	YTilt <b>nextTilt</b> ( )
py	def <b>nextTilt</b> ( )

**Returns :**

a pointer to a `YTilt` object, corresponding to a tilt sensor currently online, or a `null` pointer if there are no more tilt sensors to enumerate.

## tilt→registerTimedReportCallback() tilt.registerTimedReportCallback()

YTilt

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YTiltTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YTiltTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYTiltTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## tilt→registerValueCallback()

### tilt.registerValueCallback()

YTilt

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YTiltValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YTiltValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYTiltValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**tilt**→**set\_highestValue()****tilt**→**setHighestValue()****tilt.set\_highestValue()**

Changes the recorded maximal value observed.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YTilt <b>target set_highestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**tilt→set\_logFrequency()****YTilt****tilt→setLogFrequency()tilt.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YTilt <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**tilt→set\_logicalName()****YTilt****tilt→setLogicalName()tilt.set\_logicalName()**

Changes the logical name of the tilt sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YTilt <b>target set_logicalName newval</b>

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 tilt sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.



**tilt**→**set\_lowestValue()****YTilt****tilt**→**setLowestValue()****tilt.set\_lowestValue()**

Changes the recorded minimal value observed.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YTilt <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**tilt→set\_reportFrequency()****tilt→setReportFrequency()tilt.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

<code>js</code>	<code>function set_reportFrequency( newval)</code>
<code>nodejs</code>	<code>function set_reportFrequency( newval)</code>
<code>php</code>	<code>function set_reportFrequency( \$newval)</code>
<code>cpp</code>	<code>int set_reportFrequency( const string&amp; newval)</code>
<code>m</code>	<code>-(int) setReportFrequency : (NSString*) newval</code>
<code>pas</code>	<code>function set_reportFrequency( newval: string): integer</code>
<code>vb</code>	<code>function set_reportFrequency( ByVal newval As String) As Integer</code>
<code>cs</code>	<code>int set_reportFrequency( string newval)</code>
<code>java</code>	<code>int set_reportFrequency( String newval)</code>
<code>py</code>	<code>def set_reportFrequency( newval)</code>
<code>cmd</code>	<code>YTilt target set_reportFrequency newval</code>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**tilt→set\_resolution()****YTilt****tilt→setResolution()tilt.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YTilt <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**tilt**→**set\_userdata()****tilt**→**setUserData()****tilt.set\_userdata()**

Stores a user context provided as argument in the `userData` attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**tilt→wait\_async()****YTilt**

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.

## 3.40. Voc function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_voc.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YVoc = yoctolib.YVoc;
php	require_once('yocto_voc.php');
c++	#include "yocto_voc.h"
m	#import "yocto_voc.h"
pas	uses yocto_voc;
vb	yocto_voc.vb
cs	yocto_voc.cs
java	import com.yoctopuce.YoctoAPI.YVoc;
py	from yocto_voc import *

### Global functions

#### yFindVoc(func)

Retrieves a Volatile Organic Compound sensor for a given identifier.

#### yFirstVoc()

Starts the enumeration of Volatile Organic Compound sensors currently accessible.

### YVoc methods

#### voc→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### voc→describe()

Returns a short text that describes unambiguously the instance of the Volatile Organic Compound sensor in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### voc→get\_advertisedValue()

Returns the current value of the Volatile Organic Compound sensor (no more than 6 characters).

#### voc→get\_currentRawValue()

Returns the unrounded and uncalibrated raw value returned by the sensor.

#### voc→get\_currentValue()

Returns the current measure for the estimated VOC concentration.

#### voc→get\_errorMessage()

Returns the error message of the latest error with the Volatile Organic Compound sensor.

#### voc→get\_errorType()

Returns the numerical error code of the latest error with the Volatile Organic Compound sensor.

#### voc→get\_friendlyName()

Returns a global identifier of the Volatile Organic Compound sensor in the format MODULE\_NAME . FUNCTION\_NAME.

#### voc→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### voc→get\_functionId()

Returns the hardware identifier of the Volatile Organic Compound sensor, without reference to the module.

#### voc→get\_hardwareId()

Returns the unique hardware identifier of the Volatile Organic Compound sensor in the form `SERIAL.FUNCTIONID`.

**`voc→get_highestValue()`**

Returns the maximal value observed for the estimated VOC concentration.

**`voc→get_logFrequency()`**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**`voc→get_logicalName()`**

Returns the logical name of the Volatile Organic Compound sensor.

**`voc→get_lowestValue()`**

Returns the minimal value observed for the estimated VOC concentration.

**`voc→get_module()`**

Gets the `YModule` object for the device on which the function is located.

**`voc→get_module_async(callback, context)`**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

**`voc→get_recordedData(startTime, endTime)`**

Retrieves a `DataSet` object holding historical data for this sensor, for a specified time interval.

**`voc→get_reportFrequency()`**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**`voc→get_resolution()`**

Returns the resolution of the measured values.

**`voc→get_unit()`**

Returns the measuring unit for the estimated VOC concentration.

**`voc→get_userData()`**

Returns the value of the `userData` attribute, as previously stored using method `set_userData`.

**`voc→isOnline()`**

Checks if the Volatile Organic Compound sensor is currently reachable, without raising any error.

**`voc→isOnline_async(callback, context)`**

Checks if the Volatile Organic Compound sensor is currently reachable, without raising any error (asynchronous version).

**`voc→load(msValidity)`**

Preloads the Volatile Organic Compound sensor cache with a specified validity duration.

**`voc→loadCalibrationPoints(rawValues, refValues)`**

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

**`voc→load_async(msValidity, callback, context)`**

Preloads the Volatile Organic Compound sensor cache with a specified validity duration (asynchronous version).

**`voc→nextVoc()`**

Continues the enumeration of Volatile Organic Compound sensors started using `yFirstVoc()`.

**`voc→registerTimedReportCallback(callback)`**

Registers the callback function that is invoked on every periodic timed notification.

**`voc→registerValueCallback(callback)`**

Registers the callback function that is invoked on every change of advertised value.

**`voc→set_highestValue(newval)`**

Changes the recorded maximal value observed for the estimated VOC concentration.

### 3. Reference

---

**voc→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**voc→set\_logicalName(newval)**

Changes the logical name of the Volatile Organic Compound sensor.

**voc→set\_lowestValue(newval)**

Changes the recorded minimal value observed for the estimated VOC concentration.

**voc→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**voc→set\_resolution(newval)**

Changes the resolution of the measured physical values.

**voc→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**voc→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.



**YVoc.FindVoc()****YVoc****yFindVoc()YVoc.FindVoc()**

Retrieves a Volatile Organic Compound sensor for a given identifier.

<code>js</code>	<code>function yFindVoc( func)</code>
<code>nodejs</code>	<code>function FindVoc( func)</code>
<code>php</code>	<code>function yFindVoc( \$func)</code>
<code>cpp</code>	<code>YVoc* yFindVoc( const string&amp; func)</code>
<code>m</code>	<code>YVoc* yFindVoc( NSString* func)</code>
<code>pas</code>	<code>function yFindVoc( func: string): TYVoc</code>
<code>vb</code>	<code>function yFindVoc( ByVal func As String) As YVoc</code>
<code>cs</code>	<code>YVoc FindVoc( string func)</code>
<code>java</code>	<code>YVoc FindVoc( String func)</code>
<code>py</code>	<code>def FindVoc( func)</code>

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the Volatile Organic Compound sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YVoc.isOnline()` to test if the Volatile Organic Compound sensor is indeed online at a given time. In case of ambiguity when looking for a Volatile Organic Compound sensor 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 Volatile Organic Compound sensor

**Returns :**

a `YVoc` object allowing you to drive the Volatile Organic Compound sensor.

**YVoc.FirstVoc()****YVoc****yFirstVoc()YVoc.FirstVoc()**

Starts the enumeration of Volatile Organic Compound sensors currently accessible.

js	function <b>yFirstVoc</b> ( )
nodejs	function <b>FirstVoc</b> ( )
php	function <b>yFirstVoc</b> ( )
cpp	YVoc* <b>yFirstVoc</b> ( )
m	YVoc* <b>yFirstVoc</b> ( )
pas	function <b>yFirstVoc</b> ( ): TYVoc
vb	function <b>yFirstVoc</b> ( ) As YVoc
cs	YVoc <b>FirstVoc</b> ( )
java	YVoc <b>FirstVoc</b> ( )
py	def <b>FirstVoc</b> ( )

Use the method `YVoc.nextVoc()` to iterate on next Volatile Organic Compound sensors.

**Returns :**

a pointer to a `YVoc` object, corresponding to the first Volatile Organic Compound sensor currently online, or a `null` pointer if there are none.

**voc→calibrateFromPoints()****voc.calibrateFromPoints()****YVoc**

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                             : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                 refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YVoc target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Parameters :**

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc→describe()****voc.describe()****YVoc**

Returns a short text that describes unambiguously the instance of the Volatile Organic Compound sensor in the form `TYPE ( NAME ) =SERIAL . FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 Volatile Organic Compound sensor (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**voc**→**get\_advertisedValue()****YVoc****voc**→**advertisedValue()****voc.get\_advertisedValue()**

Returns the current value of the Volatile Organic Compound sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YVoc <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the Volatile Organic Compound sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**voc**→**get\_currentRawValue()****YVoc****voc**→**currentRawValue()****voc.get\_currentRawValue()**

Returns the unrounded and uncalibrated raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YVoc <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the unrounded and uncalibrated raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**voc**→**get\_currentValue()****YVoc****voc**→**currentValue()****voc.get\_currentValue()**

Returns the current measure for the estimated VOC concentration.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YVoc <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the estimated VOC concentration

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**voc**→**get\_errorMessage()****YVoc****voc**→**errorMessage()****voc.get\_errorMessage()**

Returns the error message of the latest error with the Volatile Organic Compound sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the Volatile Organic Compound sensor object



**voc**→**get\_errorType()****YVoc****voc**→**errorType()****voc.get\_errorType()**

Returns the numerical error code of the latest error with the Volatile Organic Compound sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the Volatile Organic Compound sensor object

**voc**→**get\_friendlyName()****YVoc****voc**→**friendlyName()****voc.get\_friendlyName()**

Returns a global identifier of the Volatile Organic Compound sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the Volatile Organic Compound sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the Volatile Organic Compound sensor (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the Volatile Organic Compound sensor using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**voc→get\_functionDescriptor()****YVoc****voc→functionDescriptor()****voc.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**voc**→**get\_functionId()****YVoc****voc**→**functionId()****voc.get\_functionId()**

Returns the hardware identifier of the Volatile Organic Compound sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the Volatile Organic Compound sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**voc**→**get\_hardwareId()****YVoc****voc**→**hardwareId()****voc.get\_hardwareId()**

Returns the unique hardware identifier of the Volatile Organic Compound sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the Volatile Organic Compound sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the Volatile Organic Compound sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**voc**→**get\_highestValue()****YVoc****voc**→**highestValue()****voc.get\_highestValue()**

Returns the maximal value observed for the estimated VOC concentration.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YVoc <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the estimated VOC concentration

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**voc→get\_logFrequency()****YVoc****voc→logFrequency()voc.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YVoc <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**voc**→**get\_logicalName()****YVoc****voc**→**logicalName()****voc.get\_logicalName()**

Returns the logical name of the Volatile Organic Compound sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YVoc <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the Volatile Organic Compound sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**voc**→**get\_lowestValue()****YVoc****voc**→**lowestValue()****voc.get\_lowestValue()**

Returns the minimal value observed for the estimated VOC concentration.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YVoc <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the estimated VOC concentration

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**voc**→**get\_module()****YVoc****voc**→**module()****voc.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**voc→get\_module\_async()**  
**voc→module\_async()**

**YVoc**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**voc**→**get\_recordedData()****voc**→**recordedData()****voc.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) recordedData : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YVoc <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

**Parameters :**

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

**Returns :**

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**voc→get\_reportFrequency()****YVoc****voc→reportFrequency()voc.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YVoc <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.

**voc**→**get\_resolution()****YVoc****voc**→**resolution()****voc.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YVoc <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**voc**→**get\_unit()****YVoc****voc**→**unit()****voc.get\_unit()**

Returns the measuring unit for the estimated VOC concentration.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YVoc <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the estimated VOC concentration

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**voc**→**get\_userData()****YVoc****voc**→**userData()****voc.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.



**voc→isOnline()****voc.isOnline()****YVoc**

Checks if the Volatile Organic Compound sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the Volatile Organic Compound sensor 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 Volatile Organic Compound sensor.

**Returns :**

`true` if the Volatile Organic Compound sensor can be reached, and `false` otherwise

**voc**→**isOnline\_async()****YVoc**

Checks if the Volatile Organic Compound sensor 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 Volatile Organic Compound sensor 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.

**voc→load()****voc.load()****YVoc**

Preloads the Volatile Organic Compound sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

## voc→loadCalibrationPoints() voc.loadCalibrationPoints()

YVoc

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                             : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)
py def loadCalibrationPoints( rawValues, refValues)
cmd YVoc target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc→load\_async()****YVoc**

Preloads the Volatile Organic Compound sensor 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 traffic 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.

**voc**→**nextVoc()****voc.nextVoc()****YVoc**

Continues the enumeration of Volatile Organic Compound sensors started using `yFirstVoc()`.

js	function <b>nextVoc</b> ( )
nodejs	function <b>nextVoc</b> ( )
php	function <b>nextVoc</b> ( )
cpp	YVoc * <b>nextVoc</b> ( )
m	-(YVoc*) <b>nextVoc</b>
pas	function <b>nextVoc</b> ( ): TYVoc
vb	function <b>nextVoc</b> ( ) As YVoc
cs	YVoc <b>nextVoc</b> ( )
java	YVoc <b>nextVoc</b> ( )
py	def <b>nextVoc</b> ( )

**Returns :**

a pointer to a `YVoc` object, corresponding to a Volatile Organic Compound sensor currently online, or a `null` pointer if there are no more Volatile Organic Compound sensors to enumerate.

## voc→registerTimedReportCallback() voc.registerTimedReportCallback()

YVoc

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YVocTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YVocTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYVocTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## voc→registerValueCallback() voc.registerValueCallback()

YVoc

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YVocValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YVocValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYVocValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.



**voc**→**set\_highestValue()****YVoc****voc**→**setHighestValue()****voc.set\_highestValue()**

Changes the recorded maximal value observed for the estimated VOC concentration.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YVoc <b>target set_highestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed for the estimated VOC concentration

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc**→**set\_logFrequency()****YVoc****voc**→**setLogFrequency()****voc.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YVoc <b>target set_logFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc→set\_logicalName()****YVoc****voc→setLogicalName()****voc.set\_logicalName()**

Changes the logical name of the Volatile Organic Compound sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YVoc <b>target set_logicalName newval</b>

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 Volatile Organic Compound sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**voc**→**set\_lowestValue()****YVoc****voc**→**setLowestValue()****voc.set\_lowestValue()**

Changes the recorded minimal value observed for the estimated VOC concentration.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YVoc <b>target set_lowestValue newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed for the estimated VOC concentration

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc→set\_reportFrequency()****YVoc****voc→setReportFrequency()****voc.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YVoc <b>target set_reportFrequency newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc**→**set\_resolution()****voc**→**setResolution()****vac.set\_resolution()**

Changes the resolution of the measured physical values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YVoc <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured physical values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voc**→**set\_userData()****YVoc****voc**→**setUserData()****voc.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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

**voc**→**wait\_async()****YVoc**

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.



## 3.41. Voltage function interface

The Yoctopuce application programming interface allows you to read an instant measure of the sensor, as well as the minimal and maximal values observed.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_voltage.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YVoltage = yoctolib.YVoltage;
php	require_once('yocto_voltage.php');
c++	#include "yocto_voltage.h"
m	#import "yocto_voltage.h"
pas	uses yocto_voltage;
vb	yocto_voltage.vb
cs	yocto_voltage.cs
java	import com.yoctopuce.YoctoAPI.YVoltage;
py	from yocto_voltage import *

### Global functions

#### yFindVoltage(func)

Retrieves a voltage sensor for a given identifier.

#### yFirstVoltage()

Starts the enumeration of voltage sensors currently accessible.

### YVoltage methods

#### voltage→calibrateFromPoints(rawValues, refValues)

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

#### voltage→describe()

Returns a short text that describes unambiguously the instance of the voltage sensor in the form `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### voltage→get\_advertisedValue()

Returns the current value of the voltage sensor (no more than 6 characters).

#### voltage→get\_currentRawValue()

Returns the uncalibrated, unrounded raw value returned by the sensor.

#### voltage→get\_currentValue()

Returns the current measure for the voltage.

#### voltage→get\_errorMessage()

Returns the error message of the latest error with the voltage sensor.

#### voltage→get\_errorType()

Returns the numerical error code of the latest error with the voltage sensor.

#### voltage→get\_friendlyName()

Returns a global identifier of the voltage sensor in the format `MODULE_NAME . FUNCTION_NAME`.

#### voltage→get\_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

#### voltage→get\_functionId()

Returns the hardware identifier of the voltage sensor, without reference to the module.

#### voltage→get\_hardwareId()

Returns the unique hardware identifier of the voltage sensor in the form `SERIAL . FUNCTIONID`.

**voltage→get\_highestValue()**

Returns the maximal value observed for the voltage.

**voltage→get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

**voltage→get\_logicalName()**

Returns the logical name of the voltage sensor.

**voltage→get\_lowestValue()**

Returns the minimal value observed for the voltage.

**voltage→get\_module()**

Gets the YModule object for the device on which the function is located.

**voltage→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**voltage→get\_recordedData(startTime, endTime)**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

**voltage→get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

**voltage→get\_resolution()**

Returns the resolution of the measured values.

**voltage→get\_unit()**

Returns the measuring unit for the voltage.

**voltage→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**voltage→isOnline()**

Checks if the voltage sensor is currently reachable, without raising any error.

**voltage→isOnline\_async(callback, context)**

Checks if the voltage sensor is currently reachable, without raising any error (asynchronous version).

**voltage→load(msValidity)**

Preloads the voltage sensor cache with a specified validity duration.

**voltage→loadCalibrationPoints(rawValues, refValues)**

Retrieves error correction data points previously entered using the method calibrateFromPoints.

**voltage→load\_async(msValidity, callback, context)**

Preloads the voltage sensor cache with a specified validity duration (asynchronous version).

**voltage→nextVoltage()**

Continues the enumeration of voltage sensors started using yFirstVoltage().

**voltage→registerTimedReportCallback(callback)**

Registers the callback function that is invoked on every periodic timed notification.

**voltage→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**voltage→set\_highestValue(newval)**

Changes the recorded maximal value observed pour the voltage.

**voltage→set\_logFrequency(newval)**

Changes the datalogger recording frequency for this function.

**voltage→set\_logicalName(newval)**

Changes the logical name of the voltage sensor.

**voltage→set\_lowestValue(newval)**

Changes the recorded minimal value observed pour the voltage.

**voltage→set\_reportFrequency(newval)**

Changes the timed value notification frequency for this function.

**voltage→set\_resolution(newval)**

Changes the resolution of the measured values.

**voltage→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**voltage→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YVoltage.FindVoltage() yFindVoltage()YVoltage.FindVoltage()

YVoltage

Retrieves a voltage sensor for a given identifier.

js	function <b>yFindVoltage</b> ( <b>func</b> )
nodejs	function <b>FindVoltage</b> ( <b>func</b> )
php	function <b>yFindVoltage</b> ( <b>\$func</b> )
c++	YVoltage* <b>yFindVoltage</b> ( const string& <b>func</b> )
m	YVoltage* <b>yFindVoltage</b> ( NSString* <b>func</b> )
pas	function <b>yFindVoltage</b> ( <b>func</b> : string): TYVoltage
vb	function <b>yFindVoltage</b> ( ByVal <b>func</b> As String) As YVoltage
cs	YVoltage <b>FindVoltage</b> ( string <b>func</b> )
java	YVoltage <b>FindVoltage</b> ( String <b>func</b> )
py	def <b>FindVoltage</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the voltage sensor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YVoltage.isOnline()` to test if the voltage sensor is indeed online at a given time. In case of ambiguity when looking for a voltage sensor 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 voltage sensor

### Returns :

a `YVoltage` object allowing you to drive the voltage sensor.

## YVoltage.FirstVoltage() yFirstVoltage()YVoltage.FirstVoltage()

## YVoltage

Starts the enumeration of voltage sensors currently accessible.

js	function <b>yFirstVoltage</b> ( )
nodejs	function <b>FirstVoltage</b> ( )
php	function <b>yFirstVoltage</b> ( )
cpp	YVoltage* <b>yFirstVoltage</b> ( )
m	YVoltage* <b>yFirstVoltage</b> ( )
pas	function <b>yFirstVoltage</b> ( ): TYVoltage
vb	function <b>yFirstVoltage</b> ( ) As YVoltage
cs	YVoltage <b>FirstVoltage</b> ( )
java	YVoltage <b>FirstVoltage</b> ( )
py	def <b>FirstVoltage</b> ( )

Use the method `YVoltage.nextVoltage( )` to iterate on next voltage sensors.

### Returns :

a pointer to a `YVoltage` object, corresponding to the first voltage sensor currently online, or a `null` pointer if there are none.

## voltage→calibrateFromPoints() voltage.calibrateFromPoints()

YVoltage

Configures error correction data points, in particular to compensate for a possible perturbation of the measure caused by an enclosure.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                             vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YVoltage target calibrateFromPoints rawValues refValues

```

It is possible to configure up to five correction points. Correction points must be provided in ascending order, and be in the range of the sensor. The device will automatically perform a linear interpolation of the error correction between specified points. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

For more information on advanced capabilities to refine the calibration of sensors, please contact [support@yoctopuce.com](mailto:support@yoctopuce.com).

### Parameters :

**rawValues** array of floating point numbers, corresponding to the raw values returned by the sensor for the correction points.

**refValues** array of floating point numbers, corresponding to the corrected values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voltage→describe()****voltage.describe()****YVoltage**

Returns a short text that describes unambiguously the instance of the voltage sensor in the form `TYPE (NAME) =SERIAL.FUNCTIONID`.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 voltage sensor (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**voltage**→**get\_advertisedValue()****YVoltage****voltage**→**advertisedValue()****voltage.get\_advertisedValue()**

Returns the current value of the voltage sensor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YVoltage <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the voltage sensor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.



**voltage**→**get\_currentRawValue()****YVoltage****voltage**→**currentRawValue()****voltage.get\_currentRawValue()**

Returns the uncalibrated, unrounded raw value returned by the sensor.

js	function <b>get_currentRawValue</b> ( )
nodejs	function <b>get_currentRawValue</b> ( )
php	function <b>get_currentRawValue</b> ( )
cpp	double <b>get_currentRawValue</b> ( )
m	-(double) currentRawValue
pas	function <b>get_currentRawValue</b> ( ): double
vb	function <b>get_currentRawValue</b> ( ) As Double
cs	double <b>get_currentRawValue</b> ( )
java	double <b>get_currentRawValue</b> ( )
py	def <b>get_currentRawValue</b> ( )
cmd	YVoltage <b>target</b> <b>get_currentRawValue</b>

**Returns :**

a floating point number corresponding to the uncalibrated, unrounded raw value returned by the sensor

On failure, throws an exception or returns Y\_CURRENTRAWVALUE\_INVALID.

**voltage**→**get\_currentValue()****YVoltage****voltage**→**currentValue()****voltage.get\_currentValue()**

Returns the current measure for the voltage.

js	function <b>get_currentValue</b> ( )
nodejs	function <b>get_currentValue</b> ( )
php	function <b>get_currentValue</b> ( )
cpp	double <b>get_currentValue</b> ( )
m	-(double) currentValue
pas	function <b>get_currentValue</b> ( ): double
vb	function <b>get_currentValue</b> ( ) As Double
cs	double <b>get_currentValue</b> ( )
java	double <b>get_currentValue</b> ( )
py	def <b>get_currentValue</b> ( )
cmd	YVoltage <b>target</b> <b>get_currentValue</b>

**Returns :**

a floating point number corresponding to the current measure for the voltage

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

---

**voltage**→**get\_errorMessage()****YVoltage****voltage**→**errorMessage()****voltage.get\_errorMessage()**

---

Returns the error message of the latest error with the voltage sensor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the voltage sensor object

**voltage**→**get\_errorType()****YVoltage****voltage**→**errorType()****voltage.get\_errorType()**

Returns the numerical error code of the latest error with the voltage sensor.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the voltage sensor object

**voltage**→**get\_friendlyName()****YVoltage****voltage**→**friendlyName()****voltage.get\_friendlyName()**

Returns a global identifier of the voltage sensor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the voltage sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the voltage sensor (for example: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the voltage sensor using logical names (ex: `MyCustomName.relay1`)

On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**voltage**→**get\_functionDescriptor()****YVoltage****voltage**→**functionDescriptor()****voltage.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**voltage**→**get\_functionId()****YVoltage****voltage**→**functionId()****voltage.get\_functionId()**

Returns the hardware identifier of the voltage sensor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the voltage sensor (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**voltage**→**get\_hardwareId()****YVoltage****voltage**→**hardwareId()****voltage.get\_hardwareId()**

Returns the unique hardware identifier of the voltage sensor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the voltage sensor. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the voltage sensor (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.



**voltage**→**get\_highestValue()****YVoltage****voltage**→**highestValue()****voltage.get\_highestValue()**

Returns the maximal value observed for the voltage.

js	function <b>get_highestValue</b> ( )
nodejs	function <b>get_highestValue</b> ( )
php	function <b>get_highestValue</b> ( )
cpp	double <b>get_highestValue</b> ( )
m	-(double) highestValue
pas	function <b>get_highestValue</b> ( ): double
vb	function <b>get_highestValue</b> ( ) As Double
cs	double <b>get_highestValue</b> ( )
java	double <b>get_highestValue</b> ( )
py	def <b>get_highestValue</b> ( )
cmd	YVoltage <b>target</b> <b>get_highestValue</b>

**Returns :**

a floating point number corresponding to the maximal value observed for the voltage

On failure, throws an exception or returns Y\_HIGHESTVALUE\_INVALID.

**voltage→get\_logFrequency()****YVoltage****voltage→logFrequency()****voltage.get\_logFrequency()**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

js	function <b>get_logFrequency</b> ( )
nodejs	function <b>get_logFrequency</b> ( )
php	function <b>get_logFrequency</b> ( )
cpp	string <b>get_logFrequency</b> ( )
m	-(NSString*) logFrequency
pas	function <b>get_logFrequency</b> ( ): string
vb	function <b>get_logFrequency</b> ( ) As String
cs	string <b>get_logFrequency</b> ( )
java	String <b>get_logFrequency</b> ( )
py	def <b>get_logFrequency</b> ( )
cmd	YVoltage <b>target</b> <b>get_logFrequency</b>

**Returns :**

a string corresponding to the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory

On failure, throws an exception or returns Y\_LOGFREQUENCY\_INVALID.

**voltage**→**get\_logicalName()****YVoltage****voltage**→**logicalName()****voltage.get\_logicalName()**

Returns the logical name of the voltage sensor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YVoltage <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the voltage sensor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**voltage**→**get\_lowestValue()****YVoltage****voltage**→**lowestValue()****voltage.get\_lowestValue()**

Returns the minimal value observed for the voltage.

js	function <b>get_lowestValue</b> ( )
nodejs	function <b>get_lowestValue</b> ( )
php	function <b>get_lowestValue</b> ( )
cpp	double <b>get_lowestValue</b> ( )
m	-(double) lowestValue
pas	function <b>get_lowestValue</b> ( ): double
vb	function <b>get_lowestValue</b> ( ) As Double
cs	double <b>get_lowestValue</b> ( )
java	double <b>get_lowestValue</b> ( )
py	def <b>get_lowestValue</b> ( )
cmd	YVoltage <b>target</b> <b>get_lowestValue</b>

**Returns :**

a floating point number corresponding to the minimal value observed for the voltage

On failure, throws an exception or returns Y\_LOWESTVALUE\_INVALID.

**voltage**→**get\_module()****YVoltage****voltage**→**module()****voltage.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**voltage**→**get\_module\_async()****YVoltage****voltage**→**module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**voltage**→**get\_recordedData()****YVoltage****voltage**→**recordedData()****voltage.get\_recordedData()**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
nodejs	function <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
php	function <b>get_recordedData</b> ( <b>\$startTime</b> , <b>\$endTime</b> )
cpp	YDataSet <b>get_recordedData</b> ( s64 <b>startTime</b> , s64 <b>endTime</b> )
m	-(YDataSet*) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
pas	function <b>get_recordedData</b> ( <b>startTime</b> : int64, <b>endTime</b> : int64): TYDataSet
vb	function <b>get_recordedData</b> ( ) As YDataSet
cs	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
java	YDataSet <b>get_recordedData</b> ( long <b>startTime</b> , long <b>endTime</b> )
py	def <b>get_recordedData</b> ( <b>startTime</b> , <b>endTime</b> )
cmd	YVoltage <b>target</b> <b>get_recordedData</b> <b>startTime</b> <b>endTime</b>

The measures will be retrieved from the data logger, which must have been turned on at the desired time. See the documentation of the DataSet class for information on how to get an overview of the recorded data, and how to load progressively a large set of measures from the data logger.

This function only works if the device uses a recent firmware, as DataSet objects are not supported by firmwares older than version 13000.

#### Parameters :

- startTime** the start of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without initial limit.
- endTime** the end of the desired measure time interval, as a Unix timestamp, i.e. the number of seconds since January 1, 1970 UTC. The special value 0 can be used to include any meaasure, without ending limit.

#### Returns :

an instance of YDataSet, providing access to historical data. Past measures can be loaded progressively using methods from the YDataSet object.

**voltage→get\_reportFrequency()****YVoltage****voltage→reportFrequency()****voltage.get\_reportFrequency()**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

js	function <b>get_reportFrequency</b> ( )
nodejs	function <b>get_reportFrequency</b> ( )
php	function <b>get_reportFrequency</b> ( )
cpp	string <b>get_reportFrequency</b> ( )
m	-(NSString*) reportFrequency
pas	function <b>get_reportFrequency</b> ( ): string
vb	function <b>get_reportFrequency</b> ( ) As String
cs	string <b>get_reportFrequency</b> ( )
java	String <b>get_reportFrequency</b> ( )
py	def <b>get_reportFrequency</b> ( )
cmd	YVoltage <b>target</b> <b>get_reportFrequency</b>

**Returns :**

a string corresponding to the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function

On failure, throws an exception or returns Y\_REPORTFREQUENCY\_INVALID.



**voltage**→**get\_resolution()****YVoltage****voltage**→**resolution()****voltage.get\_resolution()**

Returns the resolution of the measured values.

js	function <b>get_resolution</b> ( )
nodejs	function <b>get_resolution</b> ( )
php	function <b>get_resolution</b> ( )
cpp	double <b>get_resolution</b> ( )
m	-(double) resolution
pas	function <b>get_resolution</b> ( ): double
vb	function <b>get_resolution</b> ( ) As Double
cs	double <b>get_resolution</b> ( )
java	double <b>get_resolution</b> ( )
py	def <b>get_resolution</b> ( )
cmd	YVoltage <b>target</b> <b>get_resolution</b>

The resolution corresponds to the numerical precision of the measures, which is not always the same as the actual precision of the sensor.

**Returns :**

a floating point number corresponding to the resolution of the measured values

On failure, throws an exception or returns Y\_RESOLUTION\_INVALID.

**voltage**→**get\_unit()****YVoltage****voltage**→**unit()****voltage.get\_unit()**

Returns the measuring unit for the voltage.

js	function <b>get_unit</b> ( )
nodejs	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YVoltage <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the voltage

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**voltage**→**get\_userdata()****YVoltage****voltage**→**userData()****voltage.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**voltage→isOnline()****voltage.isOnline()****YVoltage**

Checks if the voltage sensor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the voltage sensor 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 voltage sensor.

**Returns :**

`true` if the voltage sensor can be reached, and `false` otherwise

**voltage→isOnline\_async()****YVoltage**

Checks if the voltage sensor 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 voltage sensor 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.

**voltage→load()****voltage.load()****YVoltage**

Preloads the voltage sensor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

## voltage→loadCalibrationPoints() voltage.loadCalibrationPoints()

YVoltage

Retrieves error correction data points previously entered using the method `calibrateFromPoints`.

```

js function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
cpp int loadCalibrationPoints( vector<double>& rawValues,
                             vector<double>& refValues)

m -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
   : (NSMutableArray*) refValues

pas function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt

vb procedure loadCalibrationPoints( )
cs int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)

py def loadCalibrationPoints( rawValues, refValues)
cmd YVoltage target loadCalibrationPoints rawValues refValues

```

### Parameters :

- rawValues** array of floating point numbers, that will be filled by the function with the raw sensor values for the correction points.
- refValues** array of floating point numbers, that will be filled by the function with the desired values for the correction points.

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voltage→load\_async()****YVoltage**

Preloads the voltage sensor 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 traffic 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.



**voltage→nextVoltage()voltage.nextVoltage()****YVoltage**

Continues the enumeration of voltage sensors started using `yFirstVoltage()`.

js	function <b>nextVoltage</b> ( )
nodejs	function <b>nextVoltage</b> ( )
php	function <b>nextVoltage</b> ( )
cpp	YVoltage * <b>nextVoltage</b> ( )
m	-(YVoltage*) <b>nextVoltage</b>
pas	function <b>nextVoltage</b> ( ): TYVoltage
vb	function <b>nextVoltage</b> ( ) As YVoltage
cs	YVoltage <b>nextVoltage</b> ( )
java	YVoltage <b>nextVoltage</b> ( )
py	def <b>nextVoltage</b> ( )

**Returns :**

a pointer to a `YVoltage` object, corresponding to a voltage sensor currently online, or a `null` pointer if there are no more voltage sensors to enumerate.

## voltage→registerTimedReportCallback() voltage.registerTimedReportCallback()

YVoltage

Registers the callback function that is invoked on every periodic timed notification.

js	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
nodejs	function <b>registerTimedReportCallback</b> ( <b>callback</b> )
php	function <b>registerTimedReportCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerTimedReportCallback</b> ( YVoltageTimedReportCallback <b>callback</b> )
m	-(int) <b>registerTimedReportCallback</b> : (YVoltageTimedReportCallback) <b>callback</b>
pas	function <b>registerTimedReportCallback</b> ( <b>callback</b> : TYVoltageTimedReportCallback): LongInt
vb	function <b>registerTimedReportCallback</b> ( ) As Integer
cs	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
java	int <b>registerTimedReportCallback</b> ( TimedReportCallback <b>callback</b> )
py	def <b>registerTimedReportCallback</b> ( <b>callback</b> )

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 an YMeasure object describing the new advertised value.

## voltage→registerValueCallback() voltage.registerValueCallback()

YVoltage

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YVoltageValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YVoltageValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYVoltageValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**voltage**→**set\_highestValue()****YVoltage****voltage**→**setHighestValue()****voltage.set\_highestValue()**

Changes the recorded maximal value observed pour the voltage.

js	function <b>set_highestValue</b> ( <b>newval</b> )
nodejs	function <b>set_highestValue</b> ( <b>newval</b> )
php	function <b>set_highestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_highestValue</b> ( double <b>newval</b> )
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_highestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_highestValue</b> ( double <b>newval</b> )
java	int <b>set_highestValue</b> ( double <b>newval</b> )
py	def <b>set_highestValue</b> ( <b>newval</b> )
cmd	YVoltage <b>target</b> <b>set_highestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded maximal value observed pour the voltage

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voltage→set\_logFrequency()****YVoltage****voltage→setLogFrequency()****voltage.set\_logFrequency()**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_logFrequency</b> ( <b>newval</b> )
php	function <b>set_logFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_logFrequency</b> ( const string& <b>newval</b> )
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_logFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logFrequency</b> ( string <b>newval</b> )
java	int <b>set_logFrequency</b> ( String <b>newval</b> )
py	def <b>set_logFrequency</b> ( <b>newval</b> )
cmd	YVoltage <b>target</b> <b>set_logFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable recording for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the datalogger recording frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voltage**→**set\_logicalName()****YVoltage****voltage**→**setLogicalName()****voltage.set\_logicalName()**

Changes the logical name of the voltage sensor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YVoltage <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 voltage sensor.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**voltage**→**set\_lowestValue()****YVoltage****voltage**→**setLowestValue()****voltage.set\_lowestValue()**

Changes the recorded minimal value observed pour the voltage.

js	function <b>set_lowestValue</b> ( <b>newval</b> )
nodejs	function <b>set_lowestValue</b> ( <b>newval</b> )
php	function <b>set_lowestValue</b> ( <b>\$newval</b> )
cpp	int <b>set_lowestValue</b> ( double <b>newval</b> )
m	-(int) setLowestValue : (double) <b>newval</b>
pas	function <b>set_lowestValue</b> ( <b>newval</b> : double): integer
vb	function <b>set_lowestValue</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_lowestValue</b> ( double <b>newval</b> )
java	int <b>set_lowestValue</b> ( double <b>newval</b> )
py	def <b>set_lowestValue</b> ( <b>newval</b> )
cmd	YVoltage <b>target</b> <b>set_lowestValue</b> <b>newval</b>

**Parameters :**

**newval** a floating point number corresponding to the recorded minimal value observed pour the voltage

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voltage→set\_reportFrequency()****YVoltage****voltage→setReportFrequency()****voltage.set\_reportFrequency()**

Changes the timed value notification frequency for this function.

js	function <b>set_reportFrequency</b> ( <b>newval</b> )
nodejs	function <b>set_reportFrequency</b> ( <b>newval</b> )
php	function <b>set_reportFrequency</b> ( <b>\$newval</b> )
cpp	int <b>set_reportFrequency</b> ( const string& <b>newval</b> )
m	-(int) setReportFrequency : (NSString*) <b>newval</b>
pas	function <b>set_reportFrequency</b> ( <b>newval</b> : string): integer
vb	function <b>set_reportFrequency</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_reportFrequency</b> ( string <b>newval</b> )
java	int <b>set_reportFrequency</b> ( String <b>newval</b> )
py	def <b>set_reportFrequency</b> ( <b>newval</b> )
cmd	YVoltage <b>target</b> <b>set_reportFrequency</b> <b>newval</b>

The frequency can be specified as samples per second, as sample per minute (for instance "15/m") or in samples per hour (eg. "4/h"). To disable timed value notifications for this function, use the value "OFF".

**Parameters :**

**newval** a string corresponding to the timed value notification frequency for this function

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**voltage**→**set\_resolution()****YVoltage****voltage**→**setResolution()****voltage.set\_resolution()**

Changes the resolution of the measured values.

js	function <b>set_resolution</b> ( <b>newval</b> )
nodejs	function <b>set_resolution</b> ( <b>newval</b> )
php	function <b>set_resolution</b> ( <b>\$newval</b> )
cpp	int <b>set_resolution</b> ( double <b>newval</b> )
m	-(int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution</b> ( <b>newval</b> : double): integer
vb	function <b>set_resolution</b> ( ByVal <b>newval</b> As Double) As Integer
cs	int <b>set_resolution</b> ( double <b>newval</b> )
java	int <b>set_resolution</b> ( double <b>newval</b> )
py	def <b>set_resolution</b> ( <b>newval</b> )
cmd	YVoltage <b>target set_resolution newval</b>

The resolution corresponds to the numerical precision when displaying value. It does not change the precision of the measure itself.

**Parameters :**

**newval** a floating point number corresponding to the resolution of the measured values

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**voltage**→**set\_userdata()****YVoltage****voltage**→**setUserData()****voltage.set\_userdata()**

Stores a user context provided as argument in the `userData` attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**voltage→wait\_async()****YVoltage**

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.

## 3.42. Voltage source function interface

Yoctopuce application programming interface allows you to control the module voltage output. You affect absolute output values or make transitions

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_vsource.js'></script>
php	require_once('yocto_vsource.php');
c++	#include "yocto_vsource.h"
m	#import "yocto_vsource.h"
pas	uses yocto_vsource;
vb	yocto_vsource.vb
cs	yocto_vsource.cs
java	import com.yoctopuce.YoctoAPI.YVSource;
py	from yocto_vsource import *

Global functions	
<b>yFindVSource(func)</b>	Retrieves a voltage source for a given identifier.
<b>yFirstVSource()</b>	Starts the enumeration of voltage sources currently accessible.
YVSource methods	
<b>vsource→describe()</b>	Returns a short text that describes the function in the form TYPE ( NAME ) =SERIAL . FUNCTIONID.
<b>vsource→get_advertisedValue()</b>	Returns the current value of the voltage source (no more than 6 characters).
<b>vsource→get_errorMessage()</b>	Returns the error message of the latest error with this function.
<b>vsource→get_errorType()</b>	Returns the numerical error code of the latest error with this function.
<b>vsource→get_extPowerFailure()</b>	Returns true if external power supply voltage is too low.
<b>vsource→get_failure()</b>	Returns true if the module is in failure mode.
<b>vsource→get_friendlyName()</b>	Returns a global identifier of the function in the format MODULE_NAME . FUNCTION_NAME.
<b>vsource→get_functionDescriptor()</b>	Returns a unique identifier of type YFUN_DESCR corresponding to the function.
<b>vsource→get_functionId()</b>	Returns the hardware identifier of the function, without reference to the module.
<b>vsource→get_hardwareId()</b>	Returns the unique hardware identifier of the function in the form SERIAL . FUNCTIONID.
<b>vsource→get_logicalName()</b>	Returns the logical name of the voltage source.
<b>vsource→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>vsource→get_module_async(callback, context)</b>	

Gets the <code>YModule</code> object for the device on which the function is located (asynchronous version).
<b><code>vsource→get_overCurrent()</code></b> Returns true if the appliance connected to the device is too greedy .
<b><code>vsource→get_overHeat()</code></b> Returns TRUE if the module is overheating.
<b><code>vsource→get_overLoad()</code></b> Returns true if the device is not able to maintaint the requested voltage output .
<b><code>vsource→get_regulationFailure()</code></b> Returns true if the voltage output is too high regarding the requested voltage .
<b><code>vsource→get_unit()</code></b> Returns the measuring unit for the voltage.
<b><code>vsource→get_userData()</code></b> Returns the value of the <code>userData</code> attribute, as previously stored using method <code>set_userData</code> .
<b><code>vsource→get_voltage()</code></b> Returns the voltage output command (mV)
<b><code>vsource→isOnline()</code></b> Checks if the function is currently reachable, without raising any error.
<b><code>vsource→isOnline_async(callback, context)</code></b> Checks if the function is currently reachable, without raising any error (asynchronous version).
<b><code>vsource→load(msValidity)</code></b> Preloads the function cache with a specified validity duration.
<b><code>vsource→load_async(msValidity, callback, context)</code></b> Preloads the function cache with a specified validity duration (asynchronous version).
<b><code>vsource→nextVSource()</code></b> Continues the enumeration of voltage sources started using <code>yFirstVSource()</code> .
<b><code>vsource→pulse(voltage, ms_duration)</code></b> Sets device output to a specific volatage, for a specified duration, then brings it automatically to 0V.
<b><code>vsource→registerValueCallback(callback)</code></b> Registers the callback function that is invoked on every change of advertised value.
<b><code>vsource→set_logicalName(newval)</code></b> Changes the logical name of the voltage source.
<b><code>vsource→set_userData(data)</code></b> Stores a user context provided as argument in the <code>userData</code> attribute of the function.
<b><code>vsource→set_voltage(newval)</code></b> Tunes the device output voltage (milliVolts).
<b><code>vsource→voltageMove(target, ms_duration)</code></b> Performs a smooth move at constant speed toward a given value.
<b><code>vsource→wait_async(callback, context)</code></b> Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**yFindVSource()** —**YVSource****YVSource.FindVSource()****YVSource.FindVSource()**

Retrieves a voltage source for a given identifier.

js	function <b>yFindVSource</b> ( <b>func</b> )
php	function <b>yFindVSource</b> ( <b>\$func</b> )
cpp	YVSource* <b>yFindVSource</b> ( const string& <b>func</b> )
m	YVSource* <b>yFindVSource</b> ( NSString* <b>func</b> )
pas	function <b>yFindVSource</b> ( <b>func</b> : string): TYVSource
vb	function <b>yFindVSource</b> ( ByVal <b>func</b> As String) As YVSource
cs	YVSource <b>FindVSource</b> ( string <b>func</b> )
java	YVSource <b>FindVSource</b> ( String <b>func</b> )
py	def <b>FindVSource</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the voltage source is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YVSource.isOnline()` to test if the voltage source is indeed online at a given time. In case of ambiguity when looking for a voltage source 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 voltage source

**Returns :**

a `YVSource` object allowing you to drive the voltage source.

**yFirstVSource()** —**YVSource****YVSource.FirstVSource()****YVSource.FirstVSource()**

Starts the enumeration of voltage sources currently accessible.

js	function <b>yFirstVSource</b> ( )
php	function <b>yFirstVSource</b> ( )
cpp	YVSource* <b>yFirstVSource</b> ( )
m	YVSource* <b>yFirstVSource</b> ( )
pas	function <b>yFirstVSource</b> ( ): TYVSource
vb	function <b>yFirstVSource</b> ( ) As YVSource
cs	YVSource <b>FirstVSource</b> ( )
java	YVSource <b>FirstVSource</b> ( )
py	def <b>FirstVSource</b> ( )

Use the method `YVSource.nextVSource( )` to iterate on next voltage sources.

**Returns :**

a pointer to a `YVSource` object, corresponding to the first voltage source currently online, or a `null` pointer if there are none.

**vsource→describe()**

Returns a short text that describes the function in the form `TYPE (NAME) =SERIAL.FUNCTIONID`.

js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )

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 function (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)



**vsource**→**get\_advertisedValue()****YVSource****vsource**→**advertisedValue()****vsource.get\_advertisedValue()**


---

Returns the current value of the voltage source (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YVSource <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the voltage source (no more than 6 characters)

On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**vsourc**→**get\_errorMessage()****YVSource****vsourc**→**errorMessage()****vsourc.errorMessage()**

Returns the error message of the latest error with this function.

js	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using this function object

**vsource**→**get\_errorType()****YVSource****vsource**→**errorType()****vsource.get\_errorType()**

Returns the numerical error code of the latest error with this function.

js	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using this function object

**vsourc**→**get\_extPowerFailure()****YVSource****vsourc**→**extPowerFailure()****vsourc.get\_extPowerFailure()**

Returns true if external power supply voltage is too low.

js	function <b>get_extPowerFailure</b> ( )
php	function <b>get_extPowerFailure</b> ( )
cpp	Y_EXTPOWERFAILURE_enum <b>get_extPowerFailure</b> ( )
m	-(Y_EXTPOWERFAILURE_enum) extPowerFailure
pas	function <b>get_extPowerFailure</b> ( ): Integer
vb	function <b>get_extPowerFailure</b> ( ) As Integer
cs	int <b>get_extPowerFailure</b> ( )
java	int <b>get_extPowerFailure</b> ( )
py	def <b>get_extPowerFailure</b> ( )
cmd	YVSource <b>target</b> <b>get_extPowerFailure</b>

**Returns :**

either Y\_EXTPOWERFAILURE\_FALSE or Y\_EXTPOWERFAILURE\_TRUE, according to true if external power supply voltage is too low

On failure, throws an exception or returns Y\_EXTPOWERFAILURE\_INVALID.

**vsource**→**get\_failure()****YVSource****vsource**→**failure()****vsource.get\_failure()**

Returns true if the module is in failure mode.

js	function <b>get_failure</b> ( )
php	function <b>get_failure</b> ( )
cpp	Y_FAILURE_enum <b>get_failure</b> ( )
m	-(Y_FAILURE_enum) failure
pas	function <b>get_failure</b> ( ): Integer
vb	function <b>get_failure</b> ( ) As Integer
cs	int <b>get_failure</b> ( )
java	int <b>get_failure</b> ( )
py	def <b>get_failure</b> ( )
cmd	YVSource <b>target</b> <b>get_failure</b>

More information can be obtained by testing `get_overheat`, `get_overcurrent` etc... When a error condition is met, the output voltage is set to zéro and cannot be changed until the `reset()` function is called.

**Returns :**

either `Y_FAILURE_FALSE` or `Y_FAILURE_TRUE`, according to true if the module is in failure mode

On failure, throws an exception or returns `Y_FAILURE_INVALID`.

**vsSource**→**get\_friendlyName()****YVSource****vsSource**→**friendlyName()**

Returns a global identifier of the function in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	virtual string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	override string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the function if they are defined, otherwise the serial number of the module and the hardware identifier of the function (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the function using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**vsourcesrc→get\_functionDescriptor()****YVSource****vsourcesrc→functionDescriptor()****vsourcesrc.get\_vsourcesrcDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**vsource**→**get\_functionId()****YVSource****vsource**→**functionId()**

Returns the hardware identifier of the function, without reference to the module.

js	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) functionId
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the function (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.



---

**vsource**→**get\_hardwareId()****YVSource****vsource**→**hardwareId()**

---

Returns the unique hardware identifier of the function in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	<b>String</b> <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the function. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the function (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**vsSource**→**get\_logicalName()****YVSource****vsSource**→**logicalName()****vsSource.get\_logicalName()**

Returns the logical name of the voltage source.

js	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YVSource <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the voltage source

On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**vsourcesource→get\_module()****YVSource****vsourcesource→module()vsourcesource.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**vsourcesrc→get\_module\_async()****YVSource****vsourcesrc→module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

```
js 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.

**vsource**→**get\_overCurrent()****YVSource****vsource**→**overCurrent()****vsource.get\_overCurrent()**

Returns true if the appliance connected to the device is too greedy .

js	function <b>get_overCurrent</b> ( )
php	function <b>get_overCurrent</b> ( )
cpp	Y_OVERCURRENT_enum <b>get_overCurrent</b> ( )
m	-(Y_OVERCURRENT_enum) overCurrent
pas	function <b>get_overCurrent</b> ( ): Integer
vb	function <b>get_overCurrent</b> ( ) As Integer
cs	int <b>get_overCurrent</b> ( )
java	int <b>get_overCurrent</b> ( )
py	def <b>get_overCurrent</b> ( )
cmd	YVSource <b>target</b> <b>get_overCurrent</b>

**Returns :**

either Y\_OVERCURRENT\_FALSE or Y\_OVERCURRENT\_TRUE, according to true if the appliance connected to the device is too greedy

On failure, throws an exception or returns Y\_OVERCURRENT\_INVALID.

**vsources**→**get\_overHeat()****YVSource****vsources**→**overHeat()****vsources.get\_overHeat()**

Returns TRUE if the module is overheating.

js	function <b>get_overHeat</b> ( )
php	function <b>get_overHeat</b> ( )
cpp	Y_OVERHEAT_enum <b>get_overHeat</b> ( )
m	-(Y_OVERHEAT_enum) overHeat
pas	function <b>get_overHeat</b> ( ): Integer
vb	function <b>get_overHeat</b> ( ) As Integer
cs	int <b>get_overHeat</b> ( )
java	int <b>get_overHeat</b> ( )
py	def <b>get_overHeat</b> ( )
cmd	YVSource <b>target</b> <b>get_overHeat</b>

**Returns :**

either Y\_OVERHEAT\_FALSE or Y\_OVERHEAT\_TRUE, according to TRUE if the module is overheating

On failure, throws an exception or returns Y\_OVERHEAT\_INVALID.

**vsource**→**get\_overLoad()****YVSource****vsource**→**overLoad()****vsource.get\_overLoad()**

Returns true if the device is not able to maintaint the requested voltage output .

js	function <b>get_overLoad</b> ( )
php	function <b>get_overLoad</b> ( )
cpp	Y_OVERLOAD_enum <b>get_overLoad</b> ( )
m	-(Y_OVERLOAD_enum) overLoad
pas	function <b>get_overLoad</b> ( ): Integer
vb	function <b>get_overLoad</b> ( ) As Integer
cs	int <b>get_overLoad</b> ( )
java	int <b>get_overLoad</b> ( )
py	def <b>get_overLoad</b> ( )
cmd	YVSource <b>target</b> <b>get_overLoad</b>

**Returns :**

either Y\_OVERLOAD\_FALSE or Y\_OVERLOAD\_TRUE, according to true if the device is not able to maintaint the requested voltage output

On failure, throws an exception or returns Y\_OVERLOAD\_INVALID.

**vsourc**→**get\_regulationFailure()****YVSource****vsourc**→**regulationFailure()****vsourc.get\_regulationFailure()**

Returns true if the voltage output is too high regarding the requested voltage .

js	function <b>get_regulationFailure</b> ( )
php	function <b>get_regulationFailure</b> ( )
cpp	Y_REGULATIONFAILURE_enum <b>get_regulationFailure</b> ( )
m	-(Y_REGULATIONFAILURE_enum) regulationFailure
pas	function <b>get_regulationFailure</b> ( ): Integer
vb	function <b>get_regulationFailure</b> ( ) As Integer
cs	int <b>get_regulationFailure</b> ( )
java	int <b>get_regulationFailure</b> ( )
py	def <b>get_regulationFailure</b> ( )
cmd	YVSource <b>target</b> <b>get_regulationFailure</b>

**Returns :**

either Y\_REGULATIONFAILURE\_FALSE or Y\_REGULATIONFAILURE\_TRUE, according to true if the voltage output is too high regarding the requested voltage

On failure, throws an exception or returns Y\_REGULATIONFAILURE\_INVALID.



**vsource**→**get\_unit()****YVSource****vsource**→**unit()****vsource.get\_unit()**

Returns the measuring unit for the voltage.

js	function <b>get_unit</b> ( )
php	function <b>get_unit</b> ( )
cpp	string <b>get_unit</b> ( )
m	-(NSString*) unit
pas	function <b>get_unit</b> ( ): string
vb	function <b>get_unit</b> ( ) As String
cs	string <b>get_unit</b> ( )
java	String <b>get_unit</b> ( )
py	def <b>get_unit</b> ( )
cmd	YVSource <b>target</b> <b>get_unit</b>

**Returns :**

a string corresponding to the measuring unit for the voltage

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**vsource**→**get\_userData()****YVSource****vsource**→**userData()****vsource.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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.

**vsource**→**get\_voltage()****YVSource****vsource**→**voltage()****vsource.get\_voltage()**

Returns the voltage output command (mV)

js	function <b>get_voltage</b> ( )
php	function <b>get_voltage</b> ( )
cpp	int <b>get_voltage</b> ( )
m	-(int) voltage
pas	function <b>get_voltage</b> ( ): LongInt
vb	function <b>get_voltage</b> ( ) As Integer
cs	int <b>get_voltage</b> ( )
java	int <b>get_voltage</b> ( )
py	def <b>get_voltage</b> ( )

**Returns :**

an integer corresponding to the voltage output command (mV)

On failure, throws an exception or returns Y\_VOLTAGE\_INVALID.

Checks if the function is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the function 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.

**Returns :**

`true` if the function can be reached, and `false` otherwise

**vsource→isOnline\_async()****YVSource**

Checks if the function is currently reachable, without raising any error (asynchronous version).

```
js function isOnline_async( callback, context)
```

If there is a cached value for the function 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 Firefox Javascript VM that does not implement context switching during blocking I/O calls.

**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.

Preloads the function cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

**vsource→load\_async()****YVSource**

Preloads the function cache with a specified validity duration (asynchronous version).

```
js 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 traffic for instance. 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 :**

- 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.

vsourcexrightarrownextVSource()vsource.nextVSource()

YVSource

Continues the enumeration of voltage sources started using yFirstVSource( ).

js

function nextVSource( )

php

function nextVSource( )

cpp

YVSource \* nextVSource( )

m

-(YVSource\*) nextVSource

pas

function nextVSource( ): TYVSource

vb

function nextVSource( ) As YVSource

cs

YVSource nextVSource( )

java

YVSource nextVSource( )

py

def nextVSource( )

Returns :

a pointer to a YVSource object, corresponding to a voltage source currently online, or a null pointer if there are no more voltage sources to enumerate.



**vsource→pulse()****vsource.pulse()****YVSource**

Sets device output to a specific volatage, for a specified duration, then brings it automatically to 0V.

js	function <b>pulse</b> ( <b>voltage</b> , <b>ms_duration</b> )
php	function <b>pulse</b> ( <b>\$voltage</b> , <b>\$ms_duration</b> )
cpp	int <b>pulse</b> ( int <b>voltage</b> , int <b>ms_duration</b> )
m	-(int) <b>pulse</b> : (int) <b>voltage</b> : (int) <b>ms_duration</b>
pas	function <b>pulse</b> ( <b>voltage</b> : integer, <b>ms_duration</b> : integer): integer
vb	function <b>pulse</b> ( ByVal <b>voltage</b> As Integer, ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>pulse</b> ( int <b>voltage</b> , int <b>ms_duration</b> )
java	int <b>pulse</b> ( int <b>voltage</b> , int <b>ms_duration</b> )
py	def <b>pulse</b> ( <b>voltage</b> , <b>ms_duration</b> )
cmd	YVSource <b>target pulse voltage ms_duration</b>

**Parameters :**

**voltage** pulse voltage, in millivolts  
**ms\_duration** pulse duration, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

## **vsources.registerValueCallback()**

YVSource

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	void <b>registerValueCallback</b> ( YDisplayUpdateCallback <b>callback</b> )
pas	procedure <b>registerValueCallback</b> ( <b>callback</b> : TGenericUpdateCallback)
vb	procedure <b>registerValueCallback</b> ( ByVal <b>callback</b> As GenericUpdateCallback)
cs	void <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
java	void <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )
m	-(void) <b>registerValueCallback</b> : (YFunctionUpdateCallback) <b>callback</b>

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.

**vsource**→**set\_logicalName()****YVSource****vsource**→**setLogicalName()****vsource.set\_logicalName()**

Changes the logical name of the voltage source.

js	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YVSource <b>target</b> <b>set_logicalName</b> <b>newval</b>

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 voltage source

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**vsourceset\_userdata()****YVSource****vsourcesetUserData()vsourceset\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : TObject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**vsource**→**set\_voltage()****YVSource****vsource**→**setVoltage()****vsource.set\_voltage()**

Tunes the device output voltage (milliVolts).

js	function <b>set_voltage</b> ( <b>newval</b> )
php	function <b>set_voltage</b> ( <b>\$newval</b> )
cpp	int <b>set_voltage</b> ( int <b>newval</b> )
m	-(int) setVoltage : (int) <b>newval</b>
pas	function <b>set_voltage</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_voltage</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_voltage</b> ( int <b>newval</b> )
java	int <b>set_voltage</b> ( int <b>newval</b> )
py	def <b>set_voltage</b> ( <b>newval</b> )
cmd	YVSource <b>target</b> <b>set_voltage</b> <b>newval</b>

**Parameters :**

**newval** an integer

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**vsource**→**voltageMove()****vsource.voltageMove()****YVSource**

Performs a smooth move at constant speed toward a given value.

js	function <b>voltageMove</b> ( <b>target</b> , <b>ms_duration</b> )
php	function <b>voltageMove</b> ( <b>\$target</b> , <b>\$ms_duration</b> )
cpp	int <b>voltageMove</b> ( int <b>target</b> , int <b>ms_duration</b> )
m	-(int) <b>voltageMove</b> : (int) <b>target</b> : (int) <b>ms_duration</b>
pas	function <b>voltageMove</b> ( <b>target</b> : integer, <b>ms_duration</b> : integer): integer
vb	function <b>voltageMove</b> ( ByVal <b>target</b> As Integer, ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>voltageMove</b> ( int <b>target</b> , int <b>ms_duration</b> )
java	int <b>voltageMove</b> ( int <b>target</b> , int <b>ms_duration</b> )
py	def <b>voltageMove</b> ( <b>target</b> , <b>ms_duration</b> )
cmd	YVSource <b>target</b> <b>voltageMove</b> <b>target</b> <b>ms_duration</b>

**Parameters :**

**target**                new output value at end of transition, in milliVolts.  
**ms\_duration**        transition duration, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**vsource**→**wait\_async()****YVSource**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

```
js 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 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 :

### 3.43. 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');
c++	#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
<b>yFindWakeUpMonitor(func)</b> Retrieves a monitor for a given identifier.
<b>yFirstWakeUpMonitor()</b> Starts the enumeration of monitors currently accessible.
YWakeupMonitor methods
<b>wakeupmonitor→describe()</b> Returns a short text that describes unambiguously the instance of the monitor in the form TYPE ( NAME ) =SERIAL . FUNCTIONID.
<b>wakeupmonitor→get_advertisedValue()</b> Returns the current value of the monitor (no more than 6 characters).
<b>wakeupmonitor→get_errorMessage()</b> Returns the error message of the latest error with the monitor.
<b>wakeupmonitor→get_errorType()</b> Returns the numerical error code of the latest error with the monitor.
<b>wakeupmonitor→get_friendlyName()</b> Returns a global identifier of the monitor in the format MODULE_NAME . FUNCTION_NAME.
<b>wakeupmonitor→get_functionDescriptor()</b> Returns a unique identifier of type YFUN_DESCR corresponding to the function.
<b>wakeupmonitor→get_functionId()</b> Returns the hardware identifier of the monitor, without reference to the module.
<b>wakeupmonitor→get_hardwareId()</b> Returns the unique hardware identifier of the monitor in the form SERIAL . FUNCTIONID.
<b>wakeupmonitor→get_logicalName()</b> Returns the logical name of the monitor.
<b>wakeupmonitor→get_module()</b> Gets the YModule object for the device on which the function is located.
<b>wakeupmonitor→get_module_async(callback, context)</b> Gets the YModule object for the device on which the function is located (asynchronous version).
<b>wakeupmonitor→get_nextWakeUp()</b>



Returns the next scheduled wake up date/time (UNIX format)
<b>wakeupmonitor→get_powerDuration()</b> Returns the maximal wake up time (in seconds) before automatically going to sleep.
<b>wakeupmonitor→get_sleepCountdown()</b> Returns the delay before the next sleep period.
<b>wakeupmonitor→get_userData()</b> Returns the value of the userData attribute, as previously stored using method set_userData.
<b>wakeupmonitor→get_wakeUpReason()</b> Returns the latest wake up reason.
<b>wakeupmonitor→get_wakeUpState()</b> Returns the current state of the monitor
<b>wakeupmonitor→isOnline()</b> Checks if the monitor is currently reachable, without raising any error.
<b>wakeupmonitor→isOnline_async(callback, context)</b> Checks if the monitor is currently reachable, without raising any error (asynchronous version).
<b>wakeupmonitor→load(msValidity)</b> Preloads the monitor cache with a specified validity duration.
<b>wakeupmonitor→load_async(msValidity, callback, context)</b> Preloads the monitor cache with a specified validity duration (asynchronous version).
<b>wakeupmonitor→nextWakeUpMonitor()</b> Continues the enumeration of monitors started using yFirstWakeUpMonitor( ).
<b>wakeupmonitor→registerValueCallback(callback)</b> Registers the callback function that is invoked on every change of advertised value.
<b>wakeupmonitor→resetSleepCountDown()</b> Resets the sleep countdown.
<b>wakeupmonitor→set_logicalName(newval)</b> Changes the logical name of the monitor.
<b>wakeupmonitor→set_nextWakeUp(newval)</b> Changes the days of the week when a wake up must take place.
<b>wakeupmonitor→set_powerDuration(newval)</b> Changes the maximal wake up time (seconds) before automatically going to sleep.
<b>wakeupmonitor→set_sleepCountdown(newval)</b> Changes the delay before the next sleep period.
<b>wakeupmonitor→set_userData(data)</b> Stores a user context provided as argument in the userData attribute of the function.
<b>wakeupmonitor→sleep(secBeforeSleep)</b> Goes to sleep until the next wake up condition is met, the RTC time must have been set before calling this function.
<b>wakeupmonitor→sleepFor(secUntilWakeUp, secBeforeSleep)</b> 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.
<b>wakeupmonitor→sleepUntil(wakeUpTime, secBeforeSleep)</b> 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.
<b>wakeupmonitor→wait_async(callback, context)</b>

### 3. Reference

---

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() yFindWakeUpMonitor() YWakeUpMonitor.FindWakeUpMonitor()

## YWakeUpMonitor

Retrieves a monitor for a given identifier.

js	function <b>yFindWakeUpMonitor</b> ( <b>func</b> )
nodejs	function <b>FindWakeUpMonitor</b> ( <b>func</b> )
php	function <b>yFindWakeUpMonitor</b> ( <b>\$func</b> )
cpp	YWakeupMonitor* <b>yFindWakeUpMonitor</b> ( const string& <b>func</b> )
m	YWakeupMonitor* <b>yFindWakeUpMonitor</b> ( NSString* <b>func</b> )
pas	function <b>yFindWakeUpMonitor</b> ( <b>func</b> : string): TYWakeUpMonitor
vb	function <b>yFindWakeUpMonitor</b> ( ByVal <b>func</b> As String) As YWakeUpMonitor
cs	YWakeupMonitor <b>FindWakeUpMonitor</b> ( string <b>func</b> )
java	YWakeupMonitor <b>FindWakeUpMonitor</b> ( String <b>func</b> )
py	def <b>FindWakeUpMonitor</b> ( <b>func</b> )

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() yFirstWakeUpMonitor() YWakeUpMonitor.FirstWakeUpMonitor()

YWakeUpMonitor

Starts the enumeration of monitors currently accessible.

js	function <b>yFirstWakeUpMonitor</b> ( )
nodejs	function <b>FirstWakeUpMonitor</b> ( )
php	function <b>yFirstWakeUpMonitor</b> ( )
cpp	YWakeUpMonitor* <b>yFirstWakeUpMonitor</b> ( )
m	YWakeUpMonitor* <b>yFirstWakeUpMonitor</b> ( )
pas	function <b>yFirstWakeUpMonitor</b> ( ): TYWakeUpMonitor
vb	function <b>yFirstWakeUpMonitor</b> ( ) As YWakeUpMonitor
cs	YWakeUpMonitor <b>FirstWakeUpMonitor</b> ( )
java	YWakeUpMonitor <b>FirstWakeUpMonitor</b> ( )
py	def <b>FirstWakeUpMonitor</b> ( )

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() wakeupmonitor.describe()

## YWakeUpMonitor

Returns a short text that describes unambiguously the instance of the monitor in the form  
TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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()****wakeupmonitor.get\_advertisedValue()**

Returns the current value of the monitor (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YWakeUpMonitor <b>target</b> <b>get_advertisedValue</b>

**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()****wakeupmonitor.get\_errorMessage()**

Returns the error message of the latest error with the monitor.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the monitor object

**wakeupmonitor→get\_errorType()****YWakeUpMonitor****wakeupmonitor→errorType()****wakeupmonitor.get\_errorType()**

Returns the numerical error code of the latest error with the monitor.

<code>js</code>	<code>function <b>get_errorType</b>( )</code>
<code>nodejs</code>	<code>function <b>get_errorType</b>( )</code>
<code>php</code>	<code>function <b>get_errorType</b>( )</code>
<code>cpp</code>	<code>YRETCODE <b>get_errorType</b>( )</code>
<code>pas</code>	<code>function <b>get_errorType</b>( ): YRETCODE</code>
<code>vb</code>	<code>function <b>get_errorType</b>( ) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE <b>get_errorType</b>( )</code>
<code>java</code>	<code>int <b>get_errorType</b>( )</code>
<code>py</code>	<code>def <b>get_errorType</b>( )</code>

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 occurred while using the monitor object



**wakeupmonitor→get\_friendlyName()****YWakeUpMonitor****wakeupmonitor→friendlyName()****wakeupmonitor.get\_friendlyName()**

Returns a global identifier of the monitor in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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()**  
**wakeupmonitor→functionDescriptor()**  
**wakeupmonitor.get\_functionDescriptor()**

**YWakeUpMonitor**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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()****wakeupmonitor.get\_functionId()**

Returns the hardware identifier of the monitor, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

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()**  
**wakeupmonitor→hardwareId()**  
**wakeupmonitor.get\_hardwareId()**

**YWakeUpMonitor**

Returns the unique hardware identifier of the monitor in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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()**  
**wakeupmonitor→logicalName()**  
**wakeupmonitor.get\_logicalName()**

**YWakeUpMonitor**

Returns the logical name of the monitor.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YWakeUpMonitor <b>target</b> <b>get_logicalName</b>

**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()****wakeupmonitor.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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()**  
**wakeupmonitor**→**module\_async()**

**YWakeUpMonitor**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**wakeupmonitor**→**get\_nextWakeUp()****YWakeUpMonitor****wakeupmonitor**→**nextWakeUp()****wakeupmonitor.get\_nextWakeUp()**

Returns the next scheduled wake up date/time (UNIX format)

js	function <b>get_nextWakeUp</b> ( )
nodejs	function <b>get_nextWakeUp</b> ( )
php	function <b>get_nextWakeUp</b> ( )
cpp	s64 <b>get_nextWakeUp</b> ( )
m	-(s64) nextWakeUp
pas	function <b>get_nextWakeUp</b> ( ): int64
vb	function <b>get_nextWakeUp</b> ( ) As Long
cs	long <b>get_nextWakeUp</b> ( )
java	long <b>get_nextWakeUp</b> ( )
py	def <b>get_nextWakeUp</b> ( )

**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()****wakeupmonitor.get\_powerDuration()**

Returns the maximal wake up time (in seconds) before automatically going to sleep.

js	function <b>get_powerDuration</b> ( )
nodejs	function <b>get_powerDuration</b> ( )
php	function <b>get_powerDuration</b> ( )
cpp	int <b>get_powerDuration</b> ( )
m	-(int) powerDuration
pas	function <b>get_powerDuration</b> ( ): LongInt
vb	function <b>get_powerDuration</b> ( ) As Integer
cs	int <b>get_powerDuration</b> ( )
java	int <b>get_powerDuration</b> ( )
py	def <b>get_powerDuration</b> ( )
cmd	YWakeUpMonitor <b>target</b> <b>get_powerDuration</b>

**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()  
wakeupmonitor→sleepCountdown()  
wakeupmonitor.get\_sleepCountdown()

YWakeUpMonitor

Returns the delay before the next sleep period.

js	function <b>get_sleepCountdown</b> ( )
nodejs	function <b>get_sleepCountdown</b> ( )
php	function <b>get_sleepCountdown</b> ( )
cpp	int <b>get_sleepCountdown</b> ( )
m	-(int) sleepCountdown
pas	function <b>get_sleepCountdown</b> ( ): LongInt
vb	function <b>get_sleepCountdown</b> ( ) As Integer
cs	int <b>get_sleepCountdown</b> ( )
java	int <b>get_sleepCountdown</b> ( )
py	def <b>get_sleepCountdown</b> ( )
cmd	YWakeUpMonitor <b>target</b> <b>get_sleepCountdown</b>

**Returns :**

an integer corresponding to the delay before the next sleep period

On failure, throws an exception or returns Y\_SLEEPDOWNDOWN\_INVALID.

**wakeupmonitor→get\_userdata()****YWakeUpMonitor****wakeupmonitor→userData()****wakeupmonitor.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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()****wakeupmonitor.get\_wakeUpReason()**

Returns the latest wake up reason.

js	function <b>get_wakeUpReason</b> ( )
nodejs	function <b>get_wakeUpReason</b> ( )
php	function <b>get_wakeUpReason</b> ( )
cpp	Y_WAKEUPREASON_enum <b>get_wakeUpReason</b> ( )
m	-(Y_WAKEUPREASON_enum) wakeUpReason
pas	function <b>get_wakeUpReason</b> ( ): Integer
vb	function <b>get_wakeUpReason</b> ( ) As Integer
cs	int <b>get_wakeUpReason</b> ( )
java	int <b>get_wakeUpReason</b> ( )
py	def <b>get_wakeUpReason</b> ( )
cmd	YWakeUpMonitor <b>target</b> <b>get_wakeUpReason</b>

**Returns :**

a value among Y\_WAKEUPREASON\_USBPOWER, Y\_WAKEUPREASON\_EXTPOWER, Y\_WAKEUPREASON\_ENDOFSLEEP, Y\_WAKEUPREASON\_EXTSIG1, Y\_WAKEUPREASON\_EXTSIG2, Y\_WAKEUPREASON\_EXTSIG3, Y\_WAKEUPREASON\_EXTSIG4, Y\_WAKEUPREASON\_SCHEDULE1, Y\_WAKEUPREASON\_SCHEDULE2, Y\_WAKEUPREASON\_SCHEDULE3, Y\_WAKEUPREASON\_SCHEDULE4, Y\_WAKEUPREASON\_SCHEDULE5 and Y\_WAKEUPREASON\_SCHEDULE6 corresponding to the latest wake up reason

On failure, throws an exception or returns Y\_WAKEUPREASON\_INVALID.

**wakeupmonitor→get\_wakeUpState()****YWakeUpMonitor****wakeupmonitor→wakeUpState()****wakeupmonitor.get\_wakeUpState()**

Returns the current state of the monitor

<code>js</code>	<code>function get_wakeUpState( )</code>
<code>nodejs</code>	<code>function get_wakeUpState( )</code>
<code>php</code>	<code>function get_wakeUpState( )</code>
<code>cpp</code>	<code>Y_WAKEUPSTATE_enum get_wakeUpState( )</code>
<code>m</code>	<code>-(Y_WAKEUPSTATE_enum) wakeUpState</code>
<code>pas</code>	<code>function get_wakeUpState( ): Integer</code>
<code>vb</code>	<code>function get_wakeUpState( ) As Integer</code>
<code>cs</code>	<code>int get_wakeUpState( )</code>
<code>java</code>	<code>int get_wakeUpState( )</code>
<code>py</code>	<code>def get_wakeUpState( )</code>

**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()****wakeupmonitor.isOnline()****YWakeUpMonitor**

Checks if the monitor is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()****wakeupmonitor.load()****YWakeUpMonitor**

Preloads the monitor cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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 traffic 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()**  
**wakeupmonitor.nextWakeUpMonitor()****YWakeUpMonitor**

Continues the enumeration of monitors started using `yFirstWakeUpMonitor()`.

js	function <b>nextWakeUpMonitor</b> ( )
nodejs	function <b>nextWakeUpMonitor</b> ( )
php	function <b>nextWakeUpMonitor</b> ( )
cpp	YWakeupMonitor * <b>nextWakeUpMonitor</b> ( )
m	-(YWakeupMonitor*) <b>nextWakeUpMonitor</b>
pas	function <b>nextWakeUpMonitor</b> ( ): TYWakeUpMonitor
vb	function <b>nextWakeUpMonitor</b> ( ) As YWakeUpMonitor
cs	YWakeupMonitor <b>nextWakeUpMonitor</b> ( )
java	YWakeupMonitor <b>nextWakeUpMonitor</b> ( )
py	def <b>nextWakeUpMonitor</b> ( )

**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() wakeupmonitor.registerValueCallback()

## YWakeUpMonitor

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YWakeUpMonitorValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YWakeUpMonitorValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYWakeUpMonitorValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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()  
wakeupmonitor.resetSleepCountDown()****YWakeUpMonitor**

Resets the sleep countdown.

js	function <b>resetSleepCountDown</b> ( )
nodejs	function <b>resetSleepCountDown</b> ( )
php	function <b>resetSleepCountDown</b> ( )
cpp	int <b>resetSleepCountDown</b> ( )
m	-(int) <b>resetSleepCountDown</b>
pas	function <b>resetSleepCountDown</b> ( ): LongInt
vb	function <b>resetSleepCountDown</b> ( ) As Integer
cs	int <b>resetSleepCountDown</b> ( )
java	int <b>resetSleepCountDown</b> ( )
py	def <b>resetSleepCountDown</b> ( )
cmd	YWakeUpMonitor <b>target resetSleepCountDown</b>

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**wakeupmonitor→set\_logicalName()**  
**wakeupmonitor→setLogicalName()**  
**wakeupmonitor.set\_logicalName()**

**YWakeUpMonitor**

Changes the logical name of the monitor.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YWakeUpMonitor <b>target</b> <b>set_logicalName</b> <b>newval</b>

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()****wakeupmonitor.set\_nextWakeUp()**

Changes the days of the week when a wake up must take place.

js	function <b>set_nextWakeUp</b> ( <b>newval</b> )
nodejs	function <b>set_nextWakeUp</b> ( <b>newval</b> )
php	function <b>set_nextWakeUp</b> ( <b>\$newval</b> )
cpp	int <b>set_nextWakeUp</b> ( s64 <b>newval</b> )
m	-(int) setNextWakeUp : (s64) <b>newval</b>
pas	function <b>set_nextWakeUp</b> ( <b>newval</b> : int64): integer
vb	function <b>set_nextWakeUp</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_nextWakeUp</b> ( long <b>newval</b> )
java	int <b>set_nextWakeUp</b> ( long <b>newval</b> )
py	def <b>set_nextWakeUp</b> ( <b>newval</b> )
cmd	YWakeUpMonitor <b>target</b> <b>set_nextWakeUp</b> <b>newval</b>

**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()****wakeupmonitor.set\_powerDuration()**

Changes the maximal wake up time (seconds) before automatically going to sleep.

js	function <b>set_powerDuration</b> ( <b>newval</b> )
nodejs	function <b>set_powerDuration</b> ( <b>newval</b> )
php	function <b>set_powerDuration</b> ( <b>\$newval</b> )
cpp	int <b>set_powerDuration</b> ( int <b>newval</b> )
m	-(int) <b>setPowerDuration</b> : (int) <b>newval</b>
pas	function <b>set_powerDuration</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_powerDuration</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_powerDuration</b> ( int <b>newval</b> )
java	int <b>set_powerDuration</b> ( int <b>newval</b> )
py	def <b>set_powerDuration</b> ( <b>newval</b> )
cmd	YWakeUpMonitor <b>target</b> <b>set_powerDuration</b> <b>newval</b>

**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()**  
**wakeupmonitor→setSleepCountdown()**  
**wakeupmonitor.set\_sleepCountdown()**

**YWakeUpMonitor**

Changes the delay before the next sleep period.

js	function <b>set_sleepCountdown</b> ( <b>newval</b> )
nodejs	function <b>set_sleepCountdown</b> ( <b>newval</b> )
php	function <b>set_sleepCountdown</b> ( <b>\$newval</b> )
cpp	int <b>set_sleepCountdown</b> ( int <b>newval</b> )
m	-(int) setSleepCountdown : (int) <b>newval</b>
pas	function <b>set_sleepCountdown</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_sleepCountdown</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_sleepCountdown</b> ( int <b>newval</b> )
java	int <b>set_sleepCountdown</b> ( int <b>newval</b> )
py	def <b>set_sleepCountdown</b> ( <b>newval</b> )
cmd	YWakeUpMonitor <b>target</b> <b>set_sleepCountdown</b> <b>newval</b>

**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()****wakeupmonitor.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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()****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 <b>sleep</b> ( <b>secBeforeSleep</b> )
nodejs	function <b>sleep</b> ( <b>secBeforeSleep</b> )
php	function <b>sleep</b> ( <b>\$secBeforeSleep</b> )
cpp	int <b>sleep</b> ( int <b>secBeforeSleep</b> )
m	-(int) <b>sleep</b> : (int) <b>secBeforeSleep</b>
pas	function <b>sleep</b> ( <b>secBeforeSleep</b> : LongInt): LongInt
vb	function <b>sleep</b> ( ) As Integer
cs	int <b>sleep</b> ( int <b>secBeforeSleep</b> )
java	int <b>sleep</b> ( int <b>secBeforeSleep</b> )
py	def <b>sleep</b> ( <b>secBeforeSleep</b> )
cmd	YWakeUpMonitor <b>target sleep secBeforeSleep</b>

**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()

### 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 <b>sleepFor</b> ( <b>secUntilWakeUp</b> , <b>secBeforeSleep</b> )
nodejs	function <b>sleepFor</b> ( <b>secUntilWakeUp</b> , <b>secBeforeSleep</b> )
php	function <b>sleepFor</b> ( <b>\$secUntilWakeUp</b> , <b>\$secBeforeSleep</b> )
cpp	int <b>sleepFor</b> ( int <b>secUntilWakeUp</b> , int <b>secBeforeSleep</b> )
m	-(int) <b>sleepFor</b> : (int) <b>secUntilWakeUp</b> : (int) <b>secBeforeSleep</b>
pas	function <b>sleepFor</b> ( <b>secUntilWakeUp</b> : LongInt, <b>secBeforeSleep</b> : LongInt): LongInt
vb	function <b>sleepFor</b> ( ) As Integer
cs	int <b>sleepFor</b> ( int <b>secUntilWakeUp</b> , int <b>secBeforeSleep</b> )
java	int <b>sleepFor</b> ( int <b>secUntilWakeUp</b> , int <b>secBeforeSleep</b> )
py	def <b>sleepFor</b> ( <b>secUntilWakeUp</b> , <b>secBeforeSleep</b> )
cmd	YWakeUpMonitor <b>target</b> <b>sleepFor</b> <b>secUntilWakeUp</b> <b>secBeforeSleep</b>

The count down before sleep can be canceled with resetSleepCountDown.

#### Parameters :

**secUntilWakeUp** sleep duration, in secondes

**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() 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 <b>sleepUntil</b> ( <b>wakeUpTime</b> , <b>secBeforeSleep</b> )
nodejs	function <b>sleepUntil</b> ( <b>wakeUpTime</b> , <b>secBeforeSleep</b> )
php	function <b>sleepUntil</b> ( <b>\$wakeUpTime</b> , <b>\$secBeforeSleep</b> )
c++	int <b>sleepUntil</b> ( int <b>wakeUpTime</b> , int <b>secBeforeSleep</b> )
m	-(int) <b>sleepUntil</b> : (int) <b>wakeUpTime</b> : (int) <b>secBeforeSleep</b>
pas	function <b>sleepUntil</b> ( <b>wakeUpTime</b> : LongInt, <b>secBeforeSleep</b> : LongInt): LongInt
vb	function <b>sleepUntil</b> ( ) As Integer
cs	int <b>sleepUntil</b> ( int <b>wakeUpTime</b> , int <b>secBeforeSleep</b> )
java	int <b>sleepUntil</b> ( int <b>wakeUpTime</b> , int <b>secBeforeSleep</b> )
py	def <b>sleepUntil</b> ( <b>wakeUpTime</b> , <b>secBeforeSleep</b> )
cmd	YWakeUpMonitor <b>target</b> <b>sleepUntil</b> <b>wakeUpTime</b> <b>secBeforeSleep</b>

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.

Forces a wake up.

js	function <b>wakeup</b> ( )
nodejs	function <b>wakeup</b> ( )
php	function <b>wakeup</b> ( )
cpp	int <b>wakeup</b> ( )
m	-(int) <b>wakeup</b>
pas	function <b>wakeup</b> ( ): LongInt
vb	function <b>wakeup</b> ( ) As Integer
cs	int <b>wakeup</b> ( )
java	int <b>wakeup</b> ( )
py	def <b>wakeup</b> ( )
cmd	YWakeUpMonitor <b>target wakeup</b>

## 3.44. 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');
c++	#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.
<b>wakeupschedule→get_minutesB()</b>	Returns the minutes in the 30-59 interval of each hour scheduled for wake up.
<b>wakeupschedule→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>wakeupschedule→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>wakeupschedule→get_monthDays()</b>	Returns the days of the month scheduled for wake up.
<b>wakeupschedule→get_months()</b>	Returns the months scheduled for wake up.
<b>wakeupschedule→get_nextOccurence()</b>	Returns the date/time (seconds) of the next wake up occurrence
<b>wakeupschedule→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set_userData.
<b>wakeupschedule→get_weekDays()</b>	Returns the days of the week scheduled for wake up.
<b>wakeupschedule→isOnline()</b>	Checks if the wake up schedule is currently reachable, without raising any error.
<b>wakeupschedule→isOnline_async(callback, context)</b>	Checks if the wake up schedule is currently reachable, without raising any error (asynchronous version).
<b>wakeupschedule→load(msValidity)</b>	Preloads the wake up schedule cache with a specified validity duration.
<b>wakeupschedule→load_async(msValidity, callback, context)</b>	Preloads the wake up schedule cache with a specified validity duration (asynchronous version).
<b>wakeupschedule→nextWakeUpSchedule()</b>	Continues the enumeration of wake up schedules started using yFirstWakeUpSchedule().
<b>wakeupschedule→registerValueCallback(callback)</b>	Registers the callback function that is invoked on every change of advertised value.
<b>wakeupschedule→set_hours(newval)</b>	Changes the hours when a wake up must take place.
<b>wakeupschedule→set_logicalName(newval)</b>	Changes the logical name of the wake up schedule.
<b>wakeupschedule→set_minutes(bitmap)</b>	Changes all the minutes where a wake up must take place.
<b>wakeupschedule→set_minutesA(newval)</b>	Changes the minutes in the 00-29 interval when a wake up must take place.
<b>wakeupschedule→set_minutesB(newval)</b>	Changes the minutes in the 30-59 interval when a wake up must take place.
<b>wakeupschedule→set_monthDays(newval)</b>	Changes the days of the month when a wake up must take place.
<b>wakeupschedule→set_months(newval)</b>	Changes the months when a wake up must take place.
<b>wakeupschedule→set_userData(data)</b>	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() yFindWakeUpSchedule() YWakeUpSchedule.FindWakeUpSchedule()

## YWakeUpSchedule

Retrieves a wake up schedule for a given identifier.

js	function <b>yFindWakeUpSchedule</b> ( <b>func</b> )
nodejs	function <b>FindWakeUpSchedule</b> ( <b>func</b> )
php	function <b>yFindWakeUpSchedule</b> ( <b>\$func</b> )
cpp	YWakeupSchedule* <b>yFindWakeUpSchedule</b> ( const string& <b>func</b> )
m	YWakeupSchedule* <b>yFindWakeUpSchedule</b> ( NSString* <b>func</b> )
pas	function <b>yFindWakeUpSchedule</b> ( <b>func</b> : string): TYWakeUpSchedule
vb	function <b>yFindWakeUpSchedule</b> ( ByVal <b>func</b> As String) As YWakeUpSchedule
cs	YWakeupSchedule <b>FindWakeUpSchedule</b> ( string <b>func</b> )
java	YWakeupSchedule <b>FindWakeUpSchedule</b> ( String <b>func</b> )
py	def <b>FindWakeUpSchedule</b> ( <b>func</b> )

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() yFirstWakeUpSchedule() YWakeUpSchedule.FirstWakeUpSchedule()

## YWakeUpSchedule

Starts the enumeration of wake up schedules currently accessible.

js	function <b>yFirstWakeUpSchedule</b> ( )
nodejs	function <b>FirstWakeUpSchedule</b> ( )
php	function <b>yFirstWakeUpSchedule</b> ( )
cpp	YWakeUpSchedule* <b>yFirstWakeUpSchedule</b> ( )
m	YWakeUpSchedule* <b>yFirstWakeUpSchedule</b> ( )
pas	function <b>yFirstWakeUpSchedule</b> ( ): TYWakeUpSchedule
vb	function <b>yFirstWakeUpSchedule</b> ( ) As YWakeUpSchedule
cs	YWakeUpSchedule <b>FirstWakeUpSchedule</b> ( )
java	YWakeUpSchedule <b>FirstWakeUpSchedule</b> ( )
py	def <b>FirstWakeUpSchedule</b> ( )

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() 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 <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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()****wakeupschedule.get\_advertisedValue()**

Returns the current value of the wake up schedule (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_advertisedValue</b>

**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()****wakeupschedule.get\_errorMessage()**

Returns the error message of the latest error with the wake up schedule.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the wake up schedule object

**wakeupschedule→get\_errorType()****YWakeUpSchedule****wakeupschedule→errorType()****wakeupschedule.get\_errorType()**

Returns the numerical error code of the latest error with the wake up schedule.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the wake up schedule object

**wakeupschedule→get\_friendlyName()****YWakeUpSchedule****wakeupschedule→friendlyName()****wakeupschedule.get\_friendlyName()**

Returns a global identifier of the wake up schedule in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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 example: `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()**  
**wakeupschedule→functionDescriptor()**  
**wakeupschedule.get\_functionDescriptor()**

**YWakeUpSchedule**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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()****wakeupschedule.get\_functionId()**

Returns the hardware identifier of the wake up schedule, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

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()****wakeupschedule.get\_hardwareId()**

Returns the unique hardware identifier of the wake up schedule in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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()****wakeupschedule.get\_hours()**

Returns the hours scheduled for wake up.

js	function <b>get_hours</b> ( )
nodejs	function <b>get_hours</b> ( )
php	function <b>get_hours</b> ( )
cpp	int <b>get_hours</b> ( )
m	-(int) hours
pas	function <b>get_hours</b> ( ): LongInt
vb	function <b>get_hours</b> ( ) As Integer
cs	int <b>get_hours</b> ( )
java	int <b>get_hours</b> ( )
py	def <b>get_hours</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_hours</b>

**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()****wakeupschedule.get\_logicalName()**

Returns the logical name of the wake up schedule.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_logicalName</b>

**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()****wakeupschedule.get\_minutes()**

Returns all the minutes of each hour that are scheduled for wake up.

js	function <b>get_minutes</b> ( )
nodejs	function <b>get_minutes</b> ( )
php	function <b>get_minutes</b> ( )
cpp	s64 <b>get_minutes</b> ( )
m	-(s64) minutes
pas	function <b>get_minutes</b> ( ): int64
vb	function <b>get_minutes</b> ( ) As Long
cs	long <b>get_minutes</b> ( )
java	long <b>get_minutes</b> ( )
py	def <b>get_minutes</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_minutes</b>

**wakeupschedule→get\_minutesA()****YWakeUpSchedule****wakeupschedule→minutesA()****wakeupschedule.get\_minutesA()**

Returns the minutes in the 00-29 interval of each hour scheduled for wake up.

js	function <b>get_minutesA</b> ( )
nodejs	function <b>get_minutesA</b> ( )
php	function <b>get_minutesA</b> ( )
cpp	int <b>get_minutesA</b> ( )
m	-(int) minutesA
pas	function <b>get_minutesA</b> ( ): LongInt
vb	function <b>get_minutesA</b> ( ) As Integer
cs	int <b>get_minutesA</b> ( )
java	int <b>get_minutesA</b> ( )
py	def <b>get_minutesA</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_minutesA</b>

**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()****wakeupschedule.get\_minutesB()**

Returns the minutes in the 30-59 interval of each hour scheduled for wake up.

js	function <b>get_minutesB</b> ( )
nodejs	function <b>get_minutesB</b> ( )
php	function <b>get_minutesB</b> ( )
cpp	int <b>get_minutesB</b> ( )
m	-(int) minutesB
pas	function <b>get_minutesB</b> ( ): LongInt
vb	function <b>get_minutesB</b> ( ) As Integer
cs	int <b>get_minutesB</b> ( )
java	int <b>get_minutesB</b> ( )
py	def <b>get_minutesB</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_minutesB</b>

**Returns :**

an integer corresponding to the minutes in the 30-59 interval of each hour scheduled for wake up

On failure, throws an exception or returns Y\_MINUTESB\_INVALID.



**wakeupschedule→get\_module()****YWakeUpSchedule****wakeupschedule→module()****wakeupschedule.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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 <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**wakeupschedule→get\_monthDays()****YWakeUpSchedule****wakeupschedule→monthDays()****wakeupschedule.get\_monthDays()**

Returns the days of the month scheduled for wake up.

js	function <b>get_monthDays</b> ( )
nodejs	function <b>get_monthDays</b> ( )
php	function <b>get_monthDays</b> ( )
cpp	int <b>get_monthDays</b> ( )
m	-(int) monthDays
pas	function <b>get_monthDays</b> ( ): LongInt
vb	function <b>get_monthDays</b> ( ) As Integer
cs	int <b>get_monthDays</b> ( )
java	int <b>get_monthDays</b> ( )
py	def <b>get_monthDays</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_monthDays</b>

**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()****wakeupschedule.get\_months()**

Returns the months scheduled for wake up.

js	function <b>get_months</b> ( )
nodejs	function <b>get_months</b> ( )
php	function <b>get_months</b> ( )
cpp	int <b>get_months</b> ( )
m	-(int) months
pas	function <b>get_months</b> ( ): LongInt
vb	function <b>get_months</b> ( ) As Integer
cs	int <b>get_months</b> ( )
java	int <b>get_months</b> ( )
py	def <b>get_months</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_months</b>

**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()****wakeupschedule.get\_nextOccurence()**

Returns the date/time (seconds) of the next wake up occurrence

js	function <b>get_nextOccurence</b> ( )
nodejs	function <b>get_nextOccurence</b> ( )
php	function <b>get_nextOccurence</b> ( )
cpp	s64 <b>get_nextOccurence</b> ( )
m	-(s64) nextOccurence
pas	function <b>get_nextOccurence</b> ( ): int64
vb	function <b>get_nextOccurence</b> ( ) As Long
cs	long <b>get_nextOccurence</b> ( )
java	long <b>get_nextOccurence</b> ( )
py	def <b>get_nextOccurence</b> ( )

**Returns :**

an integer corresponding to the date/time (seconds) of the next wake up occurrence

On failure, throws an exception or returns Y\_NEXT\_OCCURENCE\_INVALID.

**wakeupschedule**→**get\_userData()****YWakeUpSchedule****wakeupschedule**→**userData()****wakeupschedule.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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()****wakeupschedule.get\_weekDays()**

Returns the days of the week scheduled for wake up.

js	function <b>get_weekDays</b> ( )
nodejs	function <b>get_weekDays</b> ( )
php	function <b>get_weekDays</b> ( )
cpp	int <b>get_weekDays</b> ( )
m	-(int) weekDays
pas	function <b>get_weekDays</b> ( ): LongInt
vb	function <b>get_weekDays</b> ( ) As Integer
cs	int <b>get_weekDays</b> ( )
java	int <b>get_weekDays</b> ( )
py	def <b>get_weekDays</b> ( )
cmd	YWakeUpSchedule <b>target</b> <b>get_weekDays</b>

**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() wakeupschedule.isOnline()

## YWakeUpSchedule

Checks if the wake up schedule is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()**wakeupschedule.load()**YWakeUpSchedule**

Preloads the wake up schedule cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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 traffic 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() wakeupschedule.nextWakeUpSchedule()

YWakeUpSchedule

Continues the enumeration of wake up schedules started using `yFirstWakeUpSchedule()`.

js	function <b>nextWakeUpSchedule</b> ( )
nodejs	function <b>nextWakeUpSchedule</b> ( )
php	function <b>nextWakeUpSchedule</b> ( )
cpp	YWakeupSchedule * <b>nextWakeUpSchedule</b> ( )
m	-(YWakeupSchedule*) <b>nextWakeUpSchedule</b>
pas	function <b>nextWakeUpSchedule</b> ( ): TYWakeUpSchedule
vb	function <b>nextWakeUpSchedule</b> ( ) As YWakeUpSchedule
cs	YWakeupSchedule <b>nextWakeUpSchedule</b> ( )
java	YWakeupSchedule <b>nextWakeUpSchedule</b> ( )
py	def <b>nextWakeUpSchedule</b> ( )

### 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() wakeupschedule.registerValueCallback()

## YWakeUpSchedule

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YWakeUpScheduleValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YWakeUpScheduleValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYWakeUpScheduleValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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()****wakeupschedule.set\_hours()**

Changes the hours when a wake up must take place.

js	function <b>set_hours</b> ( <b>newval</b> )
nodejs	function <b>set_hours</b> ( <b>newval</b> )
php	function <b>set_hours</b> ( <b>\$newval</b> )
cpp	int <b>set_hours</b> ( int <b>newval</b> )
m	-(int) setHours : (int) <b>newval</b>
pas	function <b>set_hours</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_hours</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_hours</b> ( int <b>newval</b> )
java	int <b>set_hours</b> ( int <b>newval</b> )
py	def <b>set_hours</b> ( <b>newval</b> )
cmd	YWakeUpSchedule <b>target set_hours newval</b>

**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()****wakeupschedule.set\_logicalName()**

Changes the logical name of the wake up schedule.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YWakeUpSchedule <b>target</b> <b>set_logicalName</b> <b>newval</b>

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()****wakeupschedule.set\_minutes()**

Changes all the minutes where a wake up must take place.

js	function <b>set_minutes</b> ( <b>bitmap</b> )
nodejs	function <b>set_minutes</b> ( <b>bitmap</b> )
php	function <b>set_minutes</b> ( <b>\$bitmap</b> )
cpp	int <b>set_minutes</b> ( s64 <b>bitmap</b> )
m	-(int) setMinutes : (s64) <b>bitmap</b>
pas	function <b>set_minutes</b> ( <b>bitmap</b> : int64): LongInt
vb	function <b>set_minutes</b> ( ) As Integer
cs	int <b>set_minutes</b> ( long <b>bitmap</b> )
java	int <b>set_minutes</b> ( long <b>bitmap</b> )
py	def <b>set_minutes</b> ( <b>bitmap</b> )
cmd	YWakeUpSchedule <b>target set_minutes bitmap</b>

**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()****wakeupschedule.set\_minutesA()**

Changes the minutes in the 00-29 interval when a wake up must take place.

js	function <b>set_minutesA</b> ( <b>newval</b> )
nodejs	function <b>set_minutesA</b> ( <b>newval</b> )
php	function <b>set_minutesA</b> ( <b>\$newval</b> )
cpp	int <b>set_minutesA</b> ( int <b>newval</b> )
m	-(int) setMinutesA : (int) <b>newval</b>
pas	function <b>set_minutesA</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_minutesA</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_minutesA</b> ( int <b>newval</b> )
java	int <b>set_minutesA</b> ( int <b>newval</b> )
py	def <b>set_minutesA</b> ( <b>newval</b> )
cmd	YWakeUpSchedule <b>target</b> <b>set_minutesA</b> <b>newval</b>

**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()

wakeupschedule.set\_minutesB()

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()****wakeupschedule.set\_monthDays()**

Changes the days of the month when a wake up must take place.

js	function <b>set_monthDays</b> ( <b>newval</b> )
nodejs	function <b>set_monthDays</b> ( <b>newval</b> )
php	function <b>set_monthDays</b> ( <b>\$newval</b> )
cpp	int <b>set_monthDays</b> ( int <b>newval</b> )
m	-(int) setMonthDays : (int) <b>newval</b>
pas	function <b>set_monthDays</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_monthDays</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_monthDays</b> ( int <b>newval</b> )
java	int <b>set_monthDays</b> ( int <b>newval</b> )
py	def <b>set_monthDays</b> ( <b>newval</b> )
cmd	YWakeUpSchedule <b>target</b> <b>set_monthDays</b> <b>newval</b>

**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()****wakeupschedule.set\_months()**

Changes the months when a wake up must take place.

js	function <b>set_months</b> ( <b>newval</b> )
nodejs	function <b>set_months</b> ( <b>newval</b> )
php	function <b>set_months</b> ( <b>\$newval</b> )
cpp	int <b>set_months</b> ( int <b>newval</b> )
m	-(int) setMonths : (int) <b>newval</b>
pas	function <b>set_months</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_months</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_months</b> ( int <b>newval</b> )
java	int <b>set_months</b> ( int <b>newval</b> )
py	def <b>set_months</b> ( <b>newval</b> )
cmd	YWakeUpSchedule <b>target set_months newval</b>

**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()****wakeupschedule.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
cpp	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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()****wakeupschedule.set\_weekDays()**

Changes the days of the week when a wake up must take place.

js	function <b>set_weekDays</b> ( <b>newval</b> )
nodejs	function <b>set_weekDays</b> ( <b>newval</b> )
php	function <b>set_weekDays</b> ( <b>\$newval</b> )
cpp	int <b>set_weekDays</b> ( int <b>newval</b> )
m	-(int) setWeekDays : (int) <b>newval</b>
pas	function <b>set_weekDays</b> ( <b>newval</b> : LongInt): integer
vb	function <b>set_weekDays</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_weekDays</b> ( int <b>newval</b> )
java	int <b>set_weekDays</b> ( int <b>newval</b> )
py	def <b>set_weekDays</b> ( <b>newval</b> )
cmd	YWakeUpSchedule <b>target</b> <b>set_weekDays</b> <b>newval</b>

**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.

## 3.45. Watchdog function interface

The watchdog function works like a relay and can cause a brief power cut to an appliance after a preset delay to force this appliance to reset. The Watchdog must be called from time to time to reset the timer and prevent the appliance reset. The watchdog can be driven directly with *pulse* and *delayedpulse* methods to switch off an appliance for a given duration.

In order to use the functions described here, you should include:

js	<script type='text/javascript' src='yocto_watchdog.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWatchdog = yoctolib.YWatchdog;
php	require_once('yocto_watchdog.php');
c++	#include "yocto_watchdog.h"
m	#import "yocto_watchdog.h"
pas	uses yocto_watchdog;
vb	yocto_watchdog.vb
cs	yocto_watchdog.cs
java	import com.yoctopuce.YoctoAPI.YWatchdog;
py	from yocto_watchdog import *

### Global functions

#### yFindWatchdog(func)

Retrieves a watchdog for a given identifier.

#### yFirstWatchdog()

Starts the enumeration of watchdog currently accessible.

### YWatchdog methods

#### watchdog→delayedPulse(ms\_delay, ms\_duration)

Schedules a pulse.

#### watchdog→describe()

Returns a short text that describes unambiguously the instance of the watchdog in the form  
TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### watchdog→get\_advertisedValue()

Returns the current value of the watchdog (no more than 6 characters).

#### watchdog→get\_autoStart()

Returns the watchdog running state at module power up.

#### watchdog→get\_countdown()

Returns the number of milliseconds remaining before a pulse (delayedPulse() call) When there is no scheduled pulse, returns zero.

#### watchdog→get\_errorMessage()

Returns the error message of the latest error with the watchdog.

#### watchdog→get\_errorType()

Returns the numerical error code of the latest error with the watchdog.

#### watchdog→get\_friendlyName()

Returns a global identifier of the watchdog in the format MODULE\_NAME . FUNCTION\_NAME.

#### watchdog→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### watchdog→get\_functionId()

Returns the hardware identifier of the watchdog, without reference to the module.



**watchdog→get\_hardwareId()**

Returns the unique hardware identifier of the watchdog in the form SERIAL . FUNCTIONID.

**watchdog→get\_logicalName()**

Returns the logical name of the watchdog.

**watchdog→get\_maxTimeOnStateA()**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

**watchdog→get\_maxTimeOnStateB()**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

**watchdog→get\_module()**

Gets the YModule object for the device on which the function is located.

**watchdog→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**watchdog→get\_output()**

Returns the output state of the watchdog, when used as a simple switch (single throw).

**watchdog→get\_pulseTimer()**

Returns the number of milliseconds remaining before the watchdog is returned to idle position (state A), during a measured pulse generation.

**watchdog→get\_running()**

Returns the watchdog running state.

**watchdog→get\_state()**

Returns the state of the watchdog (A for the idle position, B for the active position).

**watchdog→get\_stateAtPowerOn()**

Returns the state of the watchdog at device startup (A for the idle position, B for the active position, UNCHANGED for no change).

**watchdog→get\_triggerDelay()**

Returns the waiting duration before a reset is automatically triggered by the watchdog, in milliseconds.

**watchdog→get\_triggerDuration()**

Returns the duration of resets caused by the watchdog, in milliseconds.

**watchdog→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set\_userData.

**watchdog→isOnline()**

Checks if the watchdog is currently reachable, without raising any error.

**watchdog→isOnline\_async(callback, context)**

Checks if the watchdog is currently reachable, without raising any error (asynchronous version).

**watchdog→load(msValidity)**

Preloads the watchdog cache with a specified validity duration.

**watchdog→load\_async(msValidity, callback, context)**

Preloads the watchdog cache with a specified validity duration (asynchronous version).

**watchdog→nextWatchdog()**

Continues the enumeration of watchdog started using yFirstWatchdog( ).

**watchdog→pulse(ms\_duration)**

Sets the relay to output B (active) for a specified duration, then brings it automatically back to output A (idle state).

**watchdog→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**watchdog→resetWatchdog()**

Resets the watchdog.

**watchdog→set\_autoStart(newval)**

Changes the watchdog running state at module power up.

**watchdog→set\_logicalName(newval)**

Changes the logical name of the watchdog.

**watchdog→set\_maxTimeOnStateA(newval)**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

**watchdog→set\_maxTimeOnStateB(newval)**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

**watchdog→set\_output(newval)**

Changes the output state of the watchdog, when used as a simple switch (single throw).

**watchdog→set\_running(newval)**

Changes the running state of the watchdog.

**watchdog→set\_state(newval)**

Changes the state of the watchdog (A for the idle position, B for the active position).

**watchdog→set\_stateAtPowerOn(newval)**

Preset the state of the watchdog at device startup (A for the idle position, B for the active position, UNCHANGED for no modification).

**watchdog→set\_triggerDelay(newval)**

Changes the waiting delay before a reset is triggered by the watchdog, in milliseconds.

**watchdog→set\_triggerDuration(newval)**

Changes the duration of resets caused by the watchdog, in milliseconds.

**watchdog→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**watchdog→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YWatchdog.FindWatchdog() yFindWatchdog()YWatchdog.FindWatchdog()

## YWatchdog

Retrieves a watchdog for a given identifier.

js	function <b>yFindWatchdog</b> ( <b>func</b> )
nodejs	function <b>FindWatchdog</b> ( <b>func</b> )
php	function <b>yFindWatchdog</b> ( <b>\$func</b> )
cpp	YWatchdog* <b>yFindWatchdog</b> ( const string& <b>func</b> )
m	YWatchdog* <b>yFindWatchdog</b> ( NSString* <b>func</b> )
pas	function <b>yFindWatchdog</b> ( <b>func</b> : string): TYWatchdog
vb	function <b>yFindWatchdog</b> ( ByVal <b>func</b> As String) As YWatchdog
cs	YWatchdog <b>FindWatchdog</b> ( string <b>func</b> )
java	YWatchdog <b>FindWatchdog</b> ( String <b>func</b> )
py	def <b>FindWatchdog</b> ( <b>func</b> )

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the watchdog is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YWatchdog.isOnline()` to test if the watchdog is indeed online at a given time. In case of ambiguity when looking for a watchdog 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 watchdog

### Returns :

a `YWatchdog` object allowing you to drive the watchdog.

**YWatchdog.FirstWatchdog()****YWatchdog****yFirstWatchdog()YWatchdog.FirstWatchdog()**

Starts the enumeration of watchdog currently accessible.

js	function <b>yFirstWatchdog</b> ( )
nodejs	function <b>FirstWatchdog</b> ( )
php	function <b>yFirstWatchdog</b> ( )
cpp	YWatchdog* <b>yFirstWatchdog</b> ( )
m	YWatchdog* <b>yFirstWatchdog</b> ( )
pas	function <b>yFirstWatchdog</b> ( ): TYWatchdog
vb	function <b>yFirstWatchdog</b> ( ) As YWatchdog
cs	YWatchdog <b>FirstWatchdog</b> ( )
java	YWatchdog <b>FirstWatchdog</b> ( )
py	def <b>FirstWatchdog</b> ( )

Use the method `YWatchdog.nextWatchdog( )` to iterate on next watchdog.

**Returns :**

a pointer to a `YWatchdog` object, corresponding to the first watchdog currently online, or a `null` pointer if there are none.

**watchdog→delayedPulse()****YWatchdog**

Schedules a pulse.

js	function <b>delayedPulse</b> ( <b>ms_delay</b> , <b>ms_duration</b> )
nodejs	function <b>delayedPulse</b> ( <b>ms_delay</b> , <b>ms_duration</b> )
php	function <b>delayedPulse</b> ( <b>\$ms_delay</b> , <b>\$ms_duration</b> )
cpp	int <b>delayedPulse</b> ( int <b>ms_delay</b> , int <b>ms_duration</b> )
m	-(int) <b>delayedPulse</b> : (int) <b>ms_delay</b> : (int) <b>ms_duration</b>
pas	function <b>delayedPulse</b> ( <b>ms_delay</b> : LongInt, <b>ms_duration</b> : LongInt): integer
vb	function <b>delayedPulse</b> ( ByVal <b>ms_delay</b> As Integer, ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>delayedPulse</b> ( int <b>ms_delay</b> , int <b>ms_duration</b> )
java	int <b>delayedPulse</b> ( int <b>ms_delay</b> , int <b>ms_duration</b> )
py	def <b>delayedPulse</b> ( <b>ms_delay</b> , <b>ms_duration</b> )
cmd	YWatchdog <b>target</b> <b>delayedPulse</b> <b>ms_delay</b> <b>ms_duration</b>

**Parameters :**

**ms\_delay** waiting time before the pulse, in milliseconds

**ms\_duration** pulse duration, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→describe()** watchdog.describe()**YWatchdog**

Returns a short text that describes unambiguously the instance of the watchdog in the form  
 TYPE (NAME) = SERIAL . FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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(BadCustomName.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 watchdog (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**watchdog→get\_advertisedValue()****YWatchdog****watchdog→advertisedValue()****watchdog.get\_advertisedValue()**

Returns the current value of the watchdog (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YWatchdog <b>target</b> <b>get_advertisedValue</b>

**Returns :**

a string corresponding to the current value of the watchdog (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**watchdog**→**get\_autoStart()****YWatchdog****watchdog**→**autoStart()****watchdog.get\_autoStart()**

Returns the watchdog runing state at module power up.

js	function <b>get_autoStart</b> ( )
nodejs	function <b>get_autoStart</b> ( )
php	function <b>get_autoStart</b> ( )
cpp	Y_AUTOSTART_enum <b>get_autoStart</b> ( )
m	-(Y_AUTOSTART_enum) autoStart
pas	function <b>get_autoStart</b> ( ): Integer
vb	function <b>get_autoStart</b> ( ) As Integer
cs	int <b>get_autoStart</b> ( )
java	int <b>get_autoStart</b> ( )
py	def <b>get_autoStart</b> ( )
cmd	YWatchdog <b>target</b> <b>get_autoStart</b>

**Returns :**

either Y\_AUTOSTART\_OFF or Y\_AUTOSTART\_ON, according to the watchdog runing state at module power up

On failure, throws an exception or returns Y\_AUTOSTART\_INVALID.



**watchdog→get\_countdown()****YWatchdog****watchdog→countdown()watchdog.get\_countdown()**

Returns the number of milliseconds remaining before a pulse (delayedPulse() call) When there is no scheduled pulse, returns zero.

js	function <b>get_countdown</b> ( )
nodejs	function <b>get_countdown</b> ( )
php	function <b>get_countdown</b> ( )
cpp	s64 <b>get_countdown</b> ( )
m	-(s64) countdown
pas	function <b>get_countdown</b> ( ): int64
vb	function <b>get_countdown</b> ( ) As Long
cs	long <b>get_countdown</b> ( )
java	long <b>get_countdown</b> ( )
py	def <b>get_countdown</b> ( )
cmd	YWatchdog <b>target</b> <b>get_countdown</b>

**Returns :**

an integer corresponding to the number of milliseconds remaining before a pulse (delayedPulse() call) When there is no scheduled pulse, returns zero

On failure, throws an exception or returns Y\_COUNTDOWN\_INVALID.

**watchdog**→**get\_errorMessage()****YWatchdog****watchdog**→**errorMessage()****watchdog.get\_errorMessage()**

Returns the error message of the latest error with the watchdog.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the watchdog object

**watchdog→get\_errorType()****YWatchdog****watchdog→errorType()watchdog.get\_errorType()**

Returns the numerical error code of the latest error with the watchdog.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the watchdog object

**watchdog→get\_friendlyName()****YWatchdog****watchdog→friendlyName()****watchdog.get\_friendlyName()**

Returns a global identifier of the watchdog in the format `MODULE_NAME . FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

The returned string uses the logical names of the module and of the watchdog if they are defined, otherwise the serial number of the module and the hardware identifier of the watchdog (for exemple: `MyCustomName.relay1`)

**Returns :**

a string that uniquely identifies the watchdog using logical names (ex: `MyCustomName.relay1`) On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

**watchdog→get\_functionDescriptor()****YWatchdog****watchdog→functionDescriptor()****watchdog.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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.

**watchdog**→**get\_functionId()****YWatchdog****watchdog**→**functionId()****watchdog.get\_functionId()**

Returns the hardware identifier of the watchdog, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

For example `relay1`

**Returns :**

a string that identifies the watchdog (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**watchdog→get\_hardwareId()****YWatchdog****watchdog→hardwareId()watchdog.get\_hardwareId()**

Returns the unique hardware identifier of the watchdog in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the watchdog. (for example `RELAYLO1-123456.relay1`)

**Returns :**

a string that uniquely identifies the watchdog (ex: `RELAYLO1-123456.relay1`) On failure, throws an exception or returns `Y_HARDWAREID_INVALID`.

**watchdog**→**get\_logicalName()****YWatchdog****watchdog**→**logicalName()****watchdog.get\_logicalName()**

Returns the logical name of the watchdog.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YWatchdog <b>target</b> <b>get_logicalName</b>

**Returns :**

a string corresponding to the logical name of the watchdog. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.



**watchdog→get\_maxTimeOnStateA()****YWatchdog****watchdog→maxTimeOnStateA()****watchdog.get\_maxTimeOnStateA()**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

js	function <b>get_maxTimeOnStateA</b> ( )
nodejs	function <b>get_maxTimeOnStateA</b> ( )
php	function <b>get_maxTimeOnStateA</b> ( )
cpp	s64 <b>get_maxTimeOnStateA</b> ( )
m	-(s64) maxTimeOnStateA
pas	function <b>get_maxTimeOnStateA</b> ( ): int64
vb	function <b>get_maxTimeOnStateA</b> ( ) As Long
cs	long <b>get_maxTimeOnStateA</b> ( )
java	long <b>get_maxTimeOnStateA</b> ( )
py	def <b>get_maxTimeOnStateA</b> ( )
cmd	YWatchdog <b>target</b> <b>get_maxTimeOnStateA</b>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEA\_INVALID.

**watchdog→get\_maxTimeOnStateB()****YWatchdog****watchdog→maxTimeOnStateB()****watchdog.get\_maxTimeOnStateB()**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

js	function <b>get_maxTimeOnStateB</b> ( )
nodejs	function <b>get_maxTimeOnStateB</b> ( )
php	function <b>get_maxTimeOnStateB</b> ( )
cpp	s64 <b>get_maxTimeOnStateB</b> ( )
m	-(s64) maxTimeOnStateB
pas	function <b>get_maxTimeOnStateB</b> ( ): int64
vb	function <b>get_maxTimeOnStateB</b> ( ) As Long
cs	long <b>get_maxTimeOnStateB</b> ( )
java	long <b>get_maxTimeOnStateB</b> ( )
py	def <b>get_maxTimeOnStateB</b> ( )
cmd	YWatchdog <b>target</b> <b>get_maxTimeOnStateB</b>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEB\_INVALID.

**watchdog→get\_module()****YWatchdog****watchdog→module()watchdog.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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

**watchdog**→**get\_module\_async()****YWatchdog****watchdog**→**module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

js	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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.

**watchdog→get\_output()****YWatchdog****watchdog→output()watchdog.get\_output()**

Returns the output state of the watchdog, when used as a simple switch (single throw).

js	function <b>get_output</b> ( )
nodejs	function <b>get_output</b> ( )
php	function <b>get_output</b> ( )
cpp	Y_OUTPUT_enum <b>get_output</b> ( )
m	-(Y_OUTPUT_enum) output
pas	function <b>get_output</b> ( ): Integer
vb	function <b>get_output</b> ( ) As Integer
cs	int <b>get_output</b> ( )
java	int <b>get_output</b> ( )
py	def <b>get_output</b> ( )
cmd	YWatchdog <b>target</b> <b>get_output</b>

**Returns :**

either Y\_OUTPUT\_OFF or Y\_OUTPUT\_ON, according to the output state of the watchdog, when used as a simple switch (single throw)

On failure, throws an exception or returns Y\_OUTPUT\_INVALID.

**watchdog→get\_pulseTimer()****YWatchdog****watchdog→pulseTimer()watchdog.get\_pulseTimer()**

Returns the number of milliseconds remaining before the watchdog is returned to idle position (state A), during a measured pulse generation.

js	function <b>get_pulseTimer</b> ( )
nodejs	function <b>get_pulseTimer</b> ( )
php	function <b>get_pulseTimer</b> ( )
cpp	s64 <b>get_pulseTimer</b> ( )
m	-(s64) pulseTimer
pas	function <b>get_pulseTimer</b> ( ): int64
vb	function <b>get_pulseTimer</b> ( ) As Long
cs	long <b>get_pulseTimer</b> ( )
java	long <b>get_pulseTimer</b> ( )
py	def <b>get_pulseTimer</b> ( )
cmd	YWatchdog <b>target</b> <b>get_pulseTimer</b>

When there is no ongoing pulse, returns zero.

**Returns :**

an integer corresponding to the number of milliseconds remaining before the watchdog is returned to idle position (state A), during a measured pulse generation

On failure, throws an exception or returns Y\_PULSETIMER\_INVALID.

**watchdog→get\_running()****YWatchdog****watchdog→running()watchdog.get\_running()**

Returns the watchdog running state.

js	function <b>get_running</b> ( )
nodejs	function <b>get_running</b> ( )
php	function <b>get_running</b> ( )
cpp	Y_RUNNING_enum <b>get_running</b> ( )
m	-(Y_RUNNING_enum) running
pas	function <b>get_running</b> ( ): Integer
vb	function <b>get_running</b> ( ) As Integer
cs	int <b>get_running</b> ( )
java	int <b>get_running</b> ( )
py	def <b>get_running</b> ( )
cmd	YWatchdog <b>target</b> <b>get_running</b>

**Returns :**

either Y\_RUNNING\_OFF or Y\_RUNNING\_ON, according to the watchdog running state

On failure, throws an exception or returns Y\_RUNNING\_INVALID.

**watchdog**→**get\_state()****YWatchdog****watchdog**→**state()****watchdog.get\_state()**

Returns the state of the watchdog (A for the idle position, B for the active position).

js	function <b>get_state</b> ( )
nodejs	function <b>get_state</b> ( )
php	function <b>get_state</b> ( )
cpp	Y_STATE_enum <b>get_state</b> ( )
m	-(Y_STATE_enum) state
pas	function <b>get_state</b> ( ): Integer
vb	function <b>get_state</b> ( ) As Integer
cs	int <b>get_state</b> ( )
java	int <b>get_state</b> ( )
py	def <b>get_state</b> ( )
cmd	YWatchdog <b>target</b> <b>get_state</b>

**Returns :**

either Y\_STATE\_A or Y\_STATE\_B, according to the state of the watchdog (A for the idle position, B for the active position)

On failure, throws an exception or returns Y\_STATE\_INVALID.



**watchdog→get\_stateAtPowerOn()****YWatchdog****watchdog→stateAtPowerOn()****watchdog.get\_stateAtPowerOn()**

Returns the state of the watchdog at device startup (A for the idle position, B for the active position, UNCHANGED for no change).

js	function <b>get_stateAtPowerOn</b> ( )
nodejs	function <b>get_stateAtPowerOn</b> ( )
php	function <b>get_stateAtPowerOn</b> ( )
cpp	Y_STATEATPOWERON_enum <b>get_stateAtPowerOn</b> ( )
m	-(Y_STATEATPOWERON_enum) stateAtPowerOn
pas	function <b>get_stateAtPowerOn</b> ( ): Integer
vb	function <b>get_stateAtPowerOn</b> ( ) As Integer
cs	int <b>get_stateAtPowerOn</b> ( )
java	int <b>get_stateAtPowerOn</b> ( )
py	def <b>get_stateAtPowerOn</b> ( )
cmd	YWatchdog <b>target</b> <b>get_stateAtPowerOn</b>

**Returns :**

a value among Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A and Y\_STATEATPOWERON\_B corresponding to the state of the watchdog at device startup (A for the idle position, B for the active position, UNCHANGED for no change)

On failure, throws an exception or returns Y\_STATEATPOWERON\_INVALID.

**watchdog→get\_triggerDelay()****YWatchdog****watchdog→triggerDelay()****watchdog.get\_triggerDelay()**

Returns the waiting duration before a reset is automatically triggered by the watchdog, in milliseconds.

js	function <b>get_triggerDelay</b> ( )
nodejs	function <b>get_triggerDelay</b> ( )
php	function <b>get_triggerDelay</b> ( )
cpp	s64 <b>get_triggerDelay</b> ( )
m	-(s64) triggerDelay
pas	function <b>get_triggerDelay</b> ( ): int64
vb	function <b>get_triggerDelay</b> ( ) As Long
cs	long <b>get_triggerDelay</b> ( )
java	long <b>get_triggerDelay</b> ( )
py	def <b>get_triggerDelay</b> ( )
cmd	YWatchdog <b>target</b> <b>get_triggerDelay</b>

**Returns :**

an integer corresponding to the waiting duration before a reset is automatically triggered by the watchdog, in milliseconds

On failure, throws an exception or returns Y\_TRIGGERDELAY\_INVALID.

**watchdog→get\_triggerDuration()****YWatchdog****watchdog→triggerDuration()****watchdog.get\_triggerDuration()**

Returns the duration of resets caused by the watchdog, in milliseconds.

js	function <b>get_triggerDuration</b> ( )
nodejs	function <b>get_triggerDuration</b> ( )
php	function <b>get_triggerDuration</b> ( )
cpp	s64 <b>get_triggerDuration</b> ( )
m	-(s64) triggerDuration
pas	function <b>get_triggerDuration</b> ( ): int64
vb	function <b>get_triggerDuration</b> ( ) As Long
cs	long <b>get_triggerDuration</b> ( )
java	long <b>get_triggerDuration</b> ( )
py	def <b>get_triggerDuration</b> ( )
cmd	YWatchdog <b>target</b> <b>get_triggerDuration</b>

**Returns :**

an integer corresponding to the duration of resets caused by the watchdog, in milliseconds

On failure, throws an exception or returns Y\_TRIGGERDURATION\_INVALID.

**watchdog**→**get\_userdata()****YWatchdog****watchdog**→**userData()****watchdog.get\_userdata()**

Returns the value of the userData attribute, as previously stored using method `set_userdata`.

js	function <b>get_userdata</b> ( )
nodejs	function <b>get_userdata</b> ( )
php	function <b>get_userdata</b> ( )
cpp	void * <b>get_userdata</b> ( )
m	-(void*) userData
pas	function <b>get_userdata</b> ( ): Tobject
vb	function <b>get_userdata</b> ( ) As Object
cs	object <b>get_userdata</b> ( )
java	Object <b>get_userdata</b> ( )
py	def <b>get_userdata</b> ( )

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.

**watchdog→isOnline()watchdog.isOnline()****YWatchdog**

Checks if the watchdog is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

If there is a cached value for the watchdog 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 watchdog.

**Returns :**

`true` if the watchdog can be reached, and `false` otherwise

**watchdog→isOnline\_async()****YWatchdog**

Checks if the watchdog 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 watchdog 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.

**watchdog→load()watchdog.load()****YWatchdog**

Preloads the watchdog cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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.

**watchdog→load\_async()****YWatchdog**

Preloads the watchdog 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 traffic 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.



## watchdog→nextWatchdog() watchdog.nextWatchdog()

## YWatchdog

Continues the enumeration of watchdog started using `yFirstWatchdog( )`.

js	function <b>nextWatchdog</b> ( )
nodejs	function <b>nextWatchdog</b> ( )
php	function <b>nextWatchdog</b> ( )
c++	YWatchdog * <b>nextWatchdog</b> ( )
m	-(YWatchdog*) <b>nextWatchdog</b>
pas	function <b>nextWatchdog</b> ( ): TYWatchdog
vb	function <b>nextWatchdog</b> ( ) As YWatchdog
cs	YWatchdog <b>nextWatchdog</b> ( )
java	YWatchdog <b>nextWatchdog</b> ( )
py	def <b>nextWatchdog</b> ( )

### Returns :

a pointer to a `YWatchdog` object, corresponding to a watchdog currently online, or a `null` pointer if there are no more watchdog to enumerate.

**watchdog→pulse()watchdog.pulse()****YWatchdog**

Sets the relay to output B (active) for a specified duration, then brings it automatically back to output A (idle state).

js	function <b>pulse</b> ( <b>ms_duration</b> )
nodejs	function <b>pulse</b> ( <b>ms_duration</b> )
php	function <b>pulse</b> ( <b>\$ms_duration</b> )
cpp	int <b>pulse</b> ( int <b>ms_duration</b> )
m	-(int) <b>pulse</b> : (int) <b>ms_duration</b>
pas	function <b>pulse</b> ( <b>ms_duration</b> : LongInt): integer
vb	function <b>pulse</b> ( ByVal <b>ms_duration</b> As Integer) As Integer
cs	int <b>pulse</b> ( int <b>ms_duration</b> )
java	int <b>pulse</b> ( int <b>ms_duration</b> )
py	def <b>pulse</b> ( <b>ms_duration</b> )
cmd	YWatchdog <b>target pulse ms_duration</b>

**Parameters :**

**ms\_duration** pulse duration, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

## watchdog→registerValueCallback() watchdog.registerValueCallback()

## YWatchdog

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YWatchdogValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YWatchdogValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYWatchdogValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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.

**watchdog→resetWatchdog()**  
**watchdog.resetWatchdog()**

YWatchdog

Resets the watchdog.

js	function <b>resetWatchdog</b> ( )
nodejs	function <b>resetWatchdog</b> ( )
php	function <b>resetWatchdog</b> ( )
cpp	int <b>resetWatchdog</b> ( )
m	-(int) <b>resetWatchdog</b>
pas	function <b>resetWatchdog</b> ( ): integer
vb	function <b>resetWatchdog</b> ( ) As Integer
cs	int <b>resetWatchdog</b> ( )
java	int <b>resetWatchdog</b> ( )
py	def <b>resetWatchdog</b> ( )
cmd	YWatchdog <b>target</b> <b>resetWatchdog</b>

When the watchdog is running, this function must be called on a regular basis to prevent the watchdog to trigger

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_autoStart()****YWatchdog****watchdog→setAutoStart()watchdog.set\_autoStart()**

Changes the watchdog runningsttae at module power up.

js	function <b>set_autoStart</b> ( <b>newval</b> )
nodejs	function <b>set_autoStart</b> ( <b>newval</b> )
php	function <b>set_autoStart</b> ( <b>\$newval</b> )
cpp	int <b>set_autoStart</b> ( Y_AUTOSTART_enum <b>newval</b> )
m	-(int) setAutoStart : (Y_AUTOSTART_enum) <b>newval</b>
pas	function <b>set_autoStart</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_autoStart</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_autoStart</b> ( int <b>newval</b> )
java	int <b>set_autoStart</b> ( int <b>newval</b> )
py	def <b>set_autoStart</b> ( <b>newval</b> )
cmd	YWatchdog <b>target set_autoStart newval</b>

Remember to call the `saveToFlash( )` method and then to reboot the module to apply this setting.

**Parameters :**

**newval** either Y\_AUTOSTART\_OFF or Y\_AUTOSTART\_ON, according to the watchdog runningsttae at module power up

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_logicalName()****YWatchdog****watchdog→setLogicalName()****watchdog.set\_logicalName()**

Changes the logical name of the watchdog.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YWatchdog <b>target set_logicalName newval</b>

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 watchdog.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**watchdog→set\_maxTimeOnStateA()****YWatchdog****watchdog→setMaxTimeOnStateA()****watchdog.set\_maxTimeOnStateA()**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

js	function <b>set_maxTimeOnStateA</b> ( <b>newval</b> )
nodejs	function <b>set_maxTimeOnStateA</b> ( <b>newval</b> )
php	function <b>set_maxTimeOnStateA</b> ( <b>\$newval</b> )
cpp	int <b>set_maxTimeOnStateA</b> ( s64 <b>newval</b> )
m	-(int) setMaxTimeOnStateA : (s64) <b>newval</b>
pas	function <b>set_maxTimeOnStateA</b> ( <b>newval</b> : int64): integer
vb	function <b>set_maxTimeOnStateA</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_maxTimeOnStateA</b> ( long <b>newval</b> )
java	int <b>set_maxTimeOnStateA</b> ( long <b>newval</b> )
py	def <b>set_maxTimeOnStateA</b> ( <b>newval</b> )
cmd	YWatchdog <b>target</b> <b>set_maxTimeOnStateA</b> <b>newval</b>

Use zero for no maximum time.

**Parameters :**

**newval** an integer

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_maxTimeOnStateB()****YWatchdog****watchdog→setMaxTimeOnStateB()****watchdog.set\_maxTimeOnStateB()**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

js	function <b>set_maxTimeOnStateB</b> ( <b>newval</b> )
nodejs	function <b>set_maxTimeOnStateB</b> ( <b>newval</b> )
php	function <b>set_maxTimeOnStateB</b> ( <b>\$newval</b> )
cpp	int <b>set_maxTimeOnStateB</b> ( s64 <b>newval</b> )
m	-(int) setMaxTimeOnStateB : (s64) <b>newval</b>
pas	function <b>set_maxTimeOnStateB</b> ( <b>newval</b> : int64): integer
vb	function <b>set_maxTimeOnStateB</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_maxTimeOnStateB</b> ( long <b>newval</b> )
java	int <b>set_maxTimeOnStateB</b> ( long <b>newval</b> )
py	def <b>set_maxTimeOnStateB</b> ( <b>newval</b> )
cmd	YWatchdog <b>target set_maxTimeOnStateB newval</b>

Use zero for no maximum time.

**Parameters :**

**newval** an integer

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.



**watchdog→set\_output()****YWatchdog****watchdog→setOutput()watchdog.set\_output()**

Changes the output state of the watchdog, when used as a simple switch (single throw).

js	function <b>set_output</b> ( <b>newval</b> )
nodejs	function <b>set_output</b> ( <b>newval</b> )
php	function <b>set_output</b> ( <b>\$newval</b> )
cpp	int <b>set_output</b> ( Y_OUTPUT_enum <b>newval</b> )
m	-(int) setOutput : (Y_OUTPUT_enum) <b>newval</b>
pas	function <b>set_output</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_output</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_output</b> ( int <b>newval</b> )
java	int <b>set_output</b> ( int <b>newval</b> )
py	def <b>set_output</b> ( <b>newval</b> )
cmd	YWatchdog <b>target set_output newval</b>

**Parameters :**

**newval** either Y\_OUTPUT\_OFF or Y\_OUTPUT\_ON, according to the output state of the watchdog, when used as a simple switch (single throw)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_running()****YWatchdog****watchdog→setRunning()watchdog.set\_running()**

Changes the running state of the watchdog.

js	function <b>set_running</b> ( <b>newval</b> )
nodejs	function <b>set_running</b> ( <b>newval</b> )
php	function <b>set_running</b> ( <b>\$newval</b> )
cpp	int <b>set_running</b> ( Y_RUNNING_enum <b>newval</b> )
m	-(int) setRunning : (Y_RUNNING_enum) <b>newval</b>
pas	function <b>set_running</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_running</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_running</b> ( int <b>newval</b> )
java	int <b>set_running</b> ( int <b>newval</b> )
py	def <b>set_running</b> ( <b>newval</b> )
cmd	YWatchdog <b>target set_running newval</b>

**Parameters :**

**newval** either Y\_RUNNING\_OFF or Y\_RUNNING\_ON, according to the running state of the watchdog

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_state()****YWatchdog****watchdog→setState()watchdog.set\_state()**

Changes the state of the watchdog (A for the idle position, B for the active position).

js	function <b>set_state</b> ( <b>newval</b> )
nodejs	function <b>set_state</b> ( <b>newval</b> )
php	function <b>set_state</b> ( <b>\$newval</b> )
cpp	int <b>set_state</b> ( Y_STATE_enum <b>newval</b> )
m	-(int) <b>setState</b> : (Y_STATE_enum) <b>newval</b>
pas	function <b>set_state</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_state</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_state</b> ( int <b>newval</b> )
java	int <b>set_state</b> ( int <b>newval</b> )
py	def <b>set_state</b> ( <b>newval</b> )
cmd	YWatchdog <b>target set_state newval</b>

**Parameters :**

**newval** either Y\_STATE\_A or Y\_STATE\_B, according to the state of the watchdog (A for the idle position, B for the active position)

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_stateAtPowerOn()****YWatchdog****watchdog→setStateAtPowerOn()****watchdog.set\_stateAtPowerOn()**

Preset the state of the watchdog at device startup (A for the idle position, B for the active position, UNCHANGED for no modification).

js	function <b>set_stateAtPowerOn</b> ( <b>newval</b> )
nodejs	function <b>set_stateAtPowerOn</b> ( <b>newval</b> )
php	function <b>set_stateAtPowerOn</b> ( <b>\$newval</b> )
cpp	int <b>set_stateAtPowerOn</b> ( Y_STATEATPOWERON_enum <b>newval</b> )
m	-(int) setStateAtPowerOn : (Y_STATEATPOWERON_enum) <b>newval</b>
pas	function <b>set_stateAtPowerOn</b> ( <b>newval</b> : Integer): integer
vb	function <b>set_stateAtPowerOn</b> ( ByVal <b>newval</b> As Integer) As Integer
cs	int <b>set_stateAtPowerOn</b> ( int <b>newval</b> )
java	int <b>set_stateAtPowerOn</b> ( int <b>newval</b> )
py	def <b>set_stateAtPowerOn</b> ( <b>newval</b> )
cmd	YWatchdog <b>target</b> <b>set_stateAtPowerOn</b> <b>newval</b>

Remember to call the matching module `saveToFlash()` method, otherwise this call will have no effect.

**Parameters :**

**newval** a value among Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A and Y\_STATEATPOWERON\_B

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_triggerDelay()**  
**watchdog→setTriggerDelay()**  
**watchdog.set\_triggerDelay()**

**YWatchdog**

Changes the waiting delay before a reset is triggered by the watchdog, in milliseconds.

js	function <b>set_triggerDelay</b> ( <b>newval</b> )
nodejs	function <b>set_triggerDelay</b> ( <b>newval</b> )
php	function <b>set_triggerDelay</b> ( <b>\$newval</b> )
cpp	int <b>set_triggerDelay</b> ( s64 <b>newval</b> )
m	-(int) setTriggerDelay : (s64) <b>newval</b>
pas	function <b>set_triggerDelay</b> ( <b>newval</b> : int64): integer
vb	function <b>set_triggerDelay</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_triggerDelay</b> ( long <b>newval</b> )
java	int <b>set_triggerDelay</b> ( long <b>newval</b> )
py	def <b>set_triggerDelay</b> ( <b>newval</b> )
cmd	YWatchdog <b>target</b> <b>set_triggerDelay</b> <b>newval</b>

#### Parameters :

**newval** an integer corresponding to the waiting delay before a reset is triggered by the watchdog, in milliseconds

#### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_triggerDuration()****YWatchdog****watchdog→setTriggerDuration()****watchdog.set\_triggerDuration()**

Changes the duration of resets caused by the watchdog, in milliseconds.

js	function <b>set_triggerDuration</b> ( <b>newval</b> )
nodejs	function <b>set_triggerDuration</b> ( <b>newval</b> )
php	function <b>set_triggerDuration</b> ( <b>\$newval</b> )
cpp	int <b>set_triggerDuration</b> ( s64 <b>newval</b> )
m	-(int) setTriggerDuration : (s64) <b>newval</b>
pas	function <b>set_triggerDuration</b> ( <b>newval</b> : int64): integer
vb	function <b>set_triggerDuration</b> ( ByVal <b>newval</b> As Long) As Integer
cs	int <b>set_triggerDuration</b> ( long <b>newval</b> )
java	int <b>set_triggerDuration</b> ( long <b>newval</b> )
py	def <b>set_triggerDuration</b> ( <b>newval</b> )
cmd	YWatchdog <b>target</b> <b>set_triggerDuration</b> <b>newval</b>

**Parameters :**

**newval** an integer corresponding to the duration of resets caused by the watchdog, in milliseconds

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**watchdog→set\_userdata()****YWatchdog****watchdog→setUserData()watchdog.set\_userdata()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userdata</b> ( <b>data</b> )
nodejs	function <b>set_userdata</b> ( <b>data</b> )
php	function <b>set_userdata</b> ( <b>\$data</b> )
c++	void <b>set_userdata</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userdata</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userdata</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userdata</b> ( object <b>data</b> )
java	void <b>set_userdata</b> ( Object <b>data</b> )
py	def <b>set_userdata</b> ( <b>data</b> )

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

**watchdog**→**wait\_async()****YWatchdog**

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.



## 3.46. 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');
c++	#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.FindWireless()

## YWireless

Retrieves a wireless lan interface for a given identifier.

js	function <b>yFindWireless</b> ( <b>func</b> )
nodejs	function <b>FindWireless</b> ( <b>func</b> )
php	function <b>yFindWireless</b> ( <b>\$func</b> )
cpp	YWireless* <b>yFindWireless</b> ( string <b>func</b> )
m	+(YWireless*) <b>yFindWireless</b> : (NSString*) <b>func</b>
pas	function <b>yFindWireless</b> ( <b>func</b> : string): TYWireless
vb	function <b>yFindWireless</b> ( ByVal <b>func</b> As String) As YWireless
cs	YWireless <b>FindWireless</b> ( string <b>func</b> )
java	YWireless <b>FindWireless</b> ( String <b>func</b> )
py	def <b>FindWireless</b> ( <b>func</b> )

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() yFirstWireless()YWireless.FirstWireless()

YWireless

Starts the enumeration of wireless lan interfaces currently accessible.

js	function <b>yFirstWireless</b> ( )
nodejs	function <b>FirstWireless</b> ( )
php	function <b>yFirstWireless</b> ( )
cpp	YWireless* <b>yFirstWireless</b> ( )
m	YWireless* <b>yFirstWireless</b> ( )
pas	function <b>yFirstWireless</b> ( ): TYWireless
vb	function <b>yFirstWireless</b> ( ) As YWireless
cs	YWireless <b>FirstWireless</b> ( )
java	YWireless <b>FirstWireless</b> ( )
py	def <b>FirstWireless</b> ( )

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()**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 <b>adhocNetwork</b> ( <b>ssid</b> , <b>securityKey</b> )
nodejs	function <b>adhocNetwork</b> ( <b>ssid</b> , <b>securityKey</b> )
php	function <b>adhocNetwork</b> ( \$ <b>ssid</b> , \$ <b>securityKey</b> )
cpp	int <b>adhocNetwork</b> ( string <b>ssid</b> , string <b>securityKey</b> )
m	-(int) <b>adhocNetwork</b> : (NSString*) <b>ssid</b> : (NSString*) <b>securityKey</b>
pas	function <b>adhocNetwork</b> ( <b>ssid</b> : string, <b>securityKey</b> : string): integer
vb	function <b>adhocNetwork</b> ( ByVal <b>ssid</b> As String, ByVal <b>securityKey</b> As String) As Integer
cs	int <b>adhocNetwork</b> ( string <b>ssid</b> , string <b>securityKey</b> )
java	int <b>adhocNetwork</b> ( String <b>ssid</b> , String <b>securityKey</b> )
py	def <b>adhocNetwork</b> ( <b>ssid</b> , <b>securityKey</b> )
cmd	YWireless <b>target</b> <b>adhocNetwork</b> <b>ssid</b> <b>securityKey</b>

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()**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 <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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()****wireless.get\_advertisedValue()**

Returns the current value of the wireless lan interface (no more than 6 characters).

js	function <b>get_advertisedValue</b> ( )
nodejs	function <b>get_advertisedValue</b> ( )
php	function <b>get_advertisedValue</b> ( )
cpp	string <b>get_advertisedValue</b> ( )
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue</b> ( ): string
vb	function <b>get_advertisedValue</b> ( ) As String
cs	string <b>get_advertisedValue</b> ( )
java	String <b>get_advertisedValue</b> ( )
py	def <b>get_advertisedValue</b> ( )
cmd	YWireless <b>target</b> <b>get_advertisedValue</b>

**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()****wireless.get\_channel()**

Returns the 802.11 channel currently used, or 0 when the selected network has not been found.

js	function <b>get_channel</b> ( )
nodejs	function <b>get_channel</b> ( )
php	function <b>get_channel</b> ( )
cpp	int <b>get_channel</b> ( )
m	-(int) channel
pas	function <b>get_channel</b> ( ): LongInt
vb	function <b>get_channel</b> ( ) As Integer
cs	int <b>get_channel</b> ( )
java	int <b>get_channel</b> ( )
py	def <b>get_channel</b> ( )
cmd	YWireless <b>target</b> <b>get_channel</b>

**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()****wireless.get\_detectedWlans()**

Returns a list of YWlanRecord objects that describe detected Wireless networks.

js	function <b>get_detectedWlans</b> ( )
nodejs	function <b>get_detectedWlans</b> ( )
php	function <b>get_detectedWlans</b> ( )
cpp	vector<YWlanRecord> <b>get_detectedWlans</b> ( )
m	-(NSMutableArray*) detectedWlans
pas	function <b>get_detectedWlans</b> ( ): TYWlanRecordArray
vb	function <b>get_detectedWlans</b> ( ) As List
cs	List<YWlanRecord> <b>get_detectedWlans</b> ( )
java	ArrayList<YWlanRecord> <b>get_detectedWlans</b> ( )
py	def <b>get_detectedWlans</b> ( )
cmd	YWireless <b>target</b> <b>get_detectedWlans</b>

This list is not updated when the module is already connected to an access 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()****wireless.get\_errorMessage()**

Returns the error message of the latest error with the wireless lan interface.

js	function <b>get_errorMessage</b> ( )
nodejs	function <b>get_errorMessage</b> ( )
php	function <b>get_errorMessage</b> ( )
cpp	string <b>get_errorMessage</b> ( )
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage</b> ( ): string
vb	function <b>get_errorMessage</b> ( ) As String
cs	string <b>get_errorMessage</b> ( )
java	String <b>get_errorMessage</b> ( )
py	def <b>get_errorMessage</b> ( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the wireless lan interface object

**wireless**→**get\_errorType()****YWireless****wireless**→**errorType()****wireless.get\_errorType()**

Returns the numerical error code of the latest error with the wireless lan interface.

js	function <b>get_errorType</b> ( )
nodejs	function <b>get_errorType</b> ( )
php	function <b>get_errorType</b> ( )
cpp	YRETCODE <b>get_errorType</b> ( )
pas	function <b>get_errorType</b> ( ): YRETCODE
vb	function <b>get_errorType</b> ( ) As YRETCODE
cs	YRETCODE <b>get_errorType</b> ( )
java	int <b>get_errorType</b> ( )
py	def <b>get_errorType</b> ( )

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 occurred while using the wireless lan interface object

**wireless**→**get\_friendlyName()****YWireless****wireless**→**friendlyName()****wireless.get\_friendlyName()**

Returns a global identifier of the wireless lan interface in the format `MODULE_NAME.FUNCTION_NAME`.

js	function <b>get_friendlyName</b> ( )
nodejs	function <b>get_friendlyName</b> ( )
php	function <b>get_friendlyName</b> ( )
cpp	string <b>get_friendlyName</b> ( )
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName</b> ( )
java	String <b>get_friendlyName</b> ( )
py	def <b>get_friendlyName</b> ( )

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 example: `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()****wireless.get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor</b> ( )
nodejs	function <b>get_functionDescriptor</b> ( )
php	function <b>get_functionDescriptor</b> ( )
cpp	YFUN_DESCR <b>get_functionDescriptor</b> ( )
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor</b> ( ): YFUN_DESCR
vb	function <b>get_functionDescriptor</b> ( ) As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor</b> ( )
java	String <b>get_functionDescriptor</b> ( )
py	def <b>get_functionDescriptor</b> ( )

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()****wireless.get\_functionId()**

Returns the hardware identifier of the wireless lan interface, without reference to the module.

js	function <b>get_functionId</b> ( )
nodejs	function <b>get_functionId</b> ( )
php	function <b>get_functionId</b> ( )
cpp	string <b>get_functionId</b> ( )
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId</b> ( ) As String
cs	string <b>get_functionId</b> ( )
java	String <b>get_functionId</b> ( )
py	def <b>get_functionId</b> ( )

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()****wireless.get\_hardwareId()**

Returns the unique hardware identifier of the wireless lan interface in the form `SERIAL.FUNCTIONID`.

js	function <b>get_hardwareId</b> ( )
nodejs	function <b>get_hardwareId</b> ( )
php	function <b>get_hardwareId</b> ( )
cpp	string <b>get_hardwareId</b> ( )
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId</b> ( ) As String
cs	string <b>get_hardwareId</b> ( )
java	String <b>get_hardwareId</b> ( )
py	def <b>get_hardwareId</b> ( )

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()****wireless.get\_linkQuality()**

Returns the link quality, expressed in percent.

js	function <b>get_linkQuality</b> ( )
nodejs	function <b>get_linkQuality</b> ( )
php	function <b>get_linkQuality</b> ( )
cpp	int <b>get_linkQuality</b> ( )
m	-(int) linkQuality
pas	function <b>get_linkQuality</b> ( ): LongInt
vb	function <b>get_linkQuality</b> ( ) As Integer
cs	int <b>get_linkQuality</b> ( )
java	int <b>get_linkQuality</b> ( )
py	def <b>get_linkQuality</b> ( )
cmd	YWireless <b>target</b> <b>get_linkQuality</b>

**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()****wireless.get\_logicalName()**

Returns the logical name of the wireless lan interface.

js	function <b>get_logicalName</b> ( )
nodejs	function <b>get_logicalName</b> ( )
php	function <b>get_logicalName</b> ( )
cpp	string <b>get_logicalName</b> ( )
m	-(NSString*) logicalName
pas	function <b>get_logicalName</b> ( ): string
vb	function <b>get_logicalName</b> ( ) As String
cs	string <b>get_logicalName</b> ( )
java	String <b>get_logicalName</b> ( )
py	def <b>get_logicalName</b> ( )
cmd	YWireless <b>target</b> <b>get_logicalName</b>

**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()****wireless.get\_message()**

Returns the latest status message from the wireless interface.

js	function <b>get_message</b> ( )
nodejs	function <b>get_message</b> ( )
php	function <b>get_message</b> ( )
cpp	string <b>get_message</b> ( )
m	-(NSString*) message
pas	function <b>get_message</b> ( ): string
vb	function <b>get_message</b> ( ) As String
cs	string <b>get_message</b> ( )
java	String <b>get_message</b> ( )
py	def <b>get_message</b> ( )
cmd	YWireless <b>target</b> <b>get_message</b>

**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()wireless.get\_module()**

Gets the YModule object for the device on which the function is located.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
c++	YModule * <b>get_module</b> ( )
m	-(YModule*) module
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	YModule <b>get_module</b> ( )
java	YModule <b>get_module</b> ( )
py	def <b>get_module</b> ( )

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 <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )
nodejs	function <b>get_module_async</b> ( <b>callback</b> , <b>context</b> )

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()****wireless.get\_security()**

Returns the security algorithm used by the selected wireless network.

js	function <b>get_security</b> ( )
nodejs	function <b>get_security</b> ( )
php	function <b>get_security</b> ( )
cpp	Y_SECURITY_enum <b>get_security</b> ( )
m	-(Y_SECURITY_enum) security
pas	function <b>get_security</b> ( ): Integer
vb	function <b>get_security</b> ( ) As Integer
cs	int <b>get_security</b> ( )
java	int <b>get_security</b> ( )
py	def <b>get_security</b> ( )
cmd	YWireless <b>target</b> <b>get_security</b>

**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()****wireless.get\_ssid()**

Returns the wireless network name (SSID).

js	function <b>get_ssid</b> ( )
nodejs	function <b>get_ssid</b> ( )
php	function <b>get_ssid</b> ( )
cpp	string <b>get_ssid</b> ( )
m	-(NSString*) ssid
pas	function <b>get_ssid</b> ( ): string
vb	function <b>get_ssid</b> ( ) As String
cs	string <b>get_ssid</b> ( )
java	String <b>get_ssid</b> ( )
py	def <b>get_ssid</b> ( )
cmd	YWireless <b>target</b> <b>get_ssid</b>

**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()****wireless.get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set_userData`.

js	function <b>get_userData</b> ( )
nodejs	function <b>get_userData</b> ( )
php	function <b>get_userData</b> ( )
cpp	void * <b>get_userData</b> ( )
m	-(void*) userData
pas	function <b>get_userData</b> ( ): Tobject
vb	function <b>get_userData</b> ( ) As Object
cs	object <b>get_userData</b> ( )
java	Object <b>get_userData</b> ( )
py	def <b>get_userData</b> ( )

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()****wireless.isOnline()****YWireless**

Checks if the wireless lan interface is currently reachable, without raising any error.

js	function <b>isOnline</b> ( )
nodejs	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()****wireless.joinNetwork()****YWireless**

Changes the configuration of the wireless lan interface to connect to an existing access point (infrastructure mode).

js	function <b>joinNetwork</b> ( <b>ssid</b> , <b>securityKey</b> )
nodejs	function <b>joinNetwork</b> ( <b>ssid</b> , <b>securityKey</b> )
php	function <b>joinNetwork</b> ( <b>\$ssid</b> , <b>\$securityKey</b> )
c++	int <b>joinNetwork</b> ( string <b>ssid</b> , string <b>securityKey</b> )
m	-(int) <b>joinNetwork</b> : (NSString*) <b>ssid</b> : (NSString*) <b>securityKey</b>
pas	function <b>joinNetwork</b> ( <b>ssid</b> : string, <b>securityKey</b> : string): integer
vb	function <b>joinNetwork</b> ( ByVal <b>ssid</b> As String, ByVal <b>securityKey</b> As String) As Integer
cs	int <b>joinNetwork</b> ( string <b>ssid</b> , string <b>securityKey</b> )
java	int <b>joinNetwork</b> ( String <b>ssid</b> , String <b>securityKey</b> )
py	def <b>joinNetwork</b> ( <b>ssid</b> , <b>securityKey</b> )
cmd	YWireless <b>target joinNetwork ssid securityKey</b>

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()****wireless.load()****YWireless**

Preloads the wireless lan interface cache with a specified validity duration.

js	function <b>load</b> ( <b>msValidity</b> )
nodejs	function <b>load</b> ( <b>msValidity</b> )
php	function <b>load</b> ( <b>\$msValidity</b> )
cpp	YRETCODE <b>load</b> ( int <b>msValidity</b> )
m	-(YRETCODE) <b>load</b> : (int) <b>msValidity</b>
pas	function <b>load</b> ( <b>msValidity</b> : integer): YRETCODE
vb	function <b>load</b> ( ByVal <b>msValidity</b> As Integer) As YRETCODE
cs	YRETCODE <b>load</b> ( int <b>msValidity</b> )
java	int <b>load</b> ( long <b>msValidity</b> )
py	def <b>load</b> ( <b>msValidity</b> )

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 traffic 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 traffic 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()****wireless.nextWireless()****YWireless**

Continues the enumeration of wireless lan interfaces started using `yFirstWireless()`.

js	function <b>nextWireless</b> ( )
nodejs	function <b>nextWireless</b> ( )
php	function <b>nextWireless</b> ( )
cpp	YWireless * <b>nextWireless</b> ( )
m	-(YWireless*) <b>nextWireless</b>
pas	function <b>nextWireless</b> ( ): TYWireless
vb	function <b>nextWireless</b> ( ) As YWireless
cs	YWireless <b>nextWireless</b> ( )
java	YWireless <b>nextWireless</b> ( )
py	def <b>nextWireless</b> ( )

**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() wireless.registerValueCallback()

YWireless

Registers the callback function that is invoked on every change of advertised value.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
nodejs	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
c++	int <b>registerValueCallback</b> ( YWirelessValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YWirelessValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYWirelessValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

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()****wireless.set\_logicalName()**

Changes the logical name of the wireless lan interface.

js	function <b>set_logicalName</b> ( <b>newval</b> )
nodejs	function <b>set_logicalName</b> ( <b>newval</b> )
php	function <b>set_logicalName</b> ( <b>\$newval</b> )
cpp	int <b>set_logicalName</b> ( const string& <b>newval</b> )
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName</b> ( <b>newval</b> : string): integer
vb	function <b>set_logicalName</b> ( ByVal <b>newval</b> As String) As Integer
cs	int <b>set_logicalName</b> ( string <b>newval</b> )
java	int <b>set_logicalName</b> ( String <b>newval</b> )
py	def <b>set_logicalName</b> ( <b>newval</b> )
cmd	YWireless <b>target set_logicalName newval</b>

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()****wireless.set\_userData()**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set_userData</b> ( <b>data</b> )
nodejs	function <b>set_userData</b> ( <b>data</b> )
php	function <b>set_userData</b> ( <b>\$data</b> )
cpp	void <b>set_userData</b> ( void* <b>data</b> )
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set_userData</b> ( <b>data</b> : Tobject)
vb	procedure <b>set_userData</b> ( ByVal <b>data</b> As Object)
cs	void <b>set_userData</b> ( object <b>data</b> )
java	void <b>set_userData</b> ( Object <b>data</b> )
py	def <b>set_userData</b> ( <b>data</b> )

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.



# Index

## A

Accelerometer 36  
adhocNetwork, YWireless 1728  
AnButton 82

## B

Blueprint 10

## C

calibrate, YLightSensor 765  
calibrateFromPoints, YAccelerometer 40  
calibrateFromPoints, YCarbonDioxide 128  
calibrateFromPoints, YCompass 204  
calibrateFromPoints, YCurrent 248  
calibrateFromPoints, YGenericSensor 557  
calibrateFromPoints, YGyro 607  
calibrateFromPoints, YHumidity 691  
calibrateFromPoints, YLightSensor 766  
calibrateFromPoints, YMagnetometer 809  
calibrateFromPoints, YPower 995  
calibrateFromPoints, YPressure 1042  
calibrateFromPoints, YQt 1154  
calibrateFromPoints, YSensor 1308  
calibrateFromPoints, YTemperature 1390  
calibrateFromPoints, YTilt 1435  
calibrateFromPoints, YVoc 1478  
calibrateFromPoints, YVoltage 1521  
callbackLogin, YNetwork 908  
cancel3DCalibration, YRefFrame 1228  
CarbonDioxide 124  
CheckLogicalName, YAPI 12  
clear, YDisplayLayer 460  
clearConsole, YDisplayLayer 461  
Clock 1193  
ColorLed 167  
Compass 200  
Configuration 1224  
consoleOut, YDisplayLayer 462  
copyLayerContent, YDisplay 412  
Current 244

## D

Data 322, 332, 345  
DataLogger 287  
delayedPulse, YDigitalIO 364  
delayedPulse, YRelay 1268  
delayedPulse, YWatchdog 1680  
describe, YAccelerometer 41  
describe, YAnButton 86  
describe, YCarbonDioxide 129  
describe, YColorLed 170  
describe, YCompass 205

describe, YCurrent 249  
describe, YDataLogger 290  
describe, YDigitalIO 365  
describe, YDisplay 413  
describe, YDualPower 494  
describe, YFiles 523  
describe, YGenericSensor 558  
describe, YGyro 608  
describe, YHubPort 661  
describe, YHumidity 692  
describe, YLed 733  
describe, YLightSensor 767  
describe, YMagnetometer 810  
describe, YModule 861  
describe, YNetwork 909  
describe, YOsControl 967  
describe, YPower 996  
describe, YPressure 1043  
describe, YPwmOutput 1085  
describe, YPwmPowerSource 1126  
describe, YQt 1155  
describe, YRealTimeClock 1196  
describe, YRefFrame 1229  
describe, YRelay 1269  
describe, YSensor 1309  
describe, YServo 1351  
describe, YTemperature 1391  
describe, YTilt 1436  
describe, YVoc 1479  
describe, YVoltage 1522  
describe, YVSource 1563  
describe, YWakeUpMonitor 1600  
describe, YWakeUpSchedule 1639  
describe, YWatchdog 1681  
describe, YWireless 1729  
Digital 360  
DisableExceptions, YAPI 13  
Display 408  
DisplayLayer 459  
download, YFiles 524  
download, YModule 862  
download\_async, YFiles 525  
drawBar, YDisplayLayer 463  
drawBitmap, YDisplayLayer 464  
drawCircle, YDisplayLayer 465  
drawDisc, YDisplayLayer 466  
drawImage, YDisplayLayer 467  
drawPixel, YDisplayLayer 468  
drawRect, YDisplayLayer 469  
drawText, YDisplayLayer 470  
dutyCycleMove, YPwmOutput 1086  
Dynamic 3

## E

EnableExceptions, YAPI 14  
EnableUSBHost, YAPI 15  
Error 7  
External 491

## F

fade, YDisplay 414  
Files 3, 520  
FindAccelerometer, YAccelerometer 38  
FindAnButton, YAnButton 84  
FindCarbonDioxide, YCarbonDioxide 126  
FindColorLed, YColorLed 168  
FindCompass, YCompass 202  
FindCurrent, YCurrent 246  
FindDataLogger, YDataLogger 288  
FindDigitalIO, YDigitalIO 362  
FindDisplay, YDisplay 410  
FindDualPower, YDualPower 492  
FindFiles, YFiles 521  
FindGenericSensor, YGenericSensor 555  
FindGyro, YGyro 605  
FindHubPort, YHubPort 659  
FindHumidity, YHumidity 689  
FindLed, YLed 731  
FindLightSensor, YLightSensor 763  
FindMagnetometer, YMagnetometer 807  
FindModule, YModule 859  
FindNetwork, YNetwork 906  
FindOsControl, YOsControl 965  
FindPower, YPower 993  
FindPressure, YPressure 1040  
FindPwmOutput, YPwmOutput 1083  
FindPwmPowerSource, YPwmPowerSource 1124  
FindQt, YQt 1152  
FindRealTimeClock, YRealTimeClock 1194  
FindRefFrame, YRefFrame 1226  
FindRelay, YRelay 1266  
FindSensor, YSensor 1306  
FindServo, YServo 1349  
FindTemperature, YTemperature 1388  
FindTilt, YTilt 1433  
FindVoc, YVoc 1476  
FindVoltage, YVoltage 1519  
FindVSource, YVSource 1561  
FindWakeUpMonitor, YWakeUpMonitor 1598  
FindWakeUpSchedule, YWakeUpSchedule 1637  
FindWatchdog, YWatchdog 1678  
FindWireless, YWireless 1726  
FirstAccelerometer, YAccelerometer 39  
FirstAnButton, YAnButton 85  
FirstCarbonDioxide, YCarbonDioxide 127  
FirstColorLed, YColorLed 169  
FirstCompass, YCompass 203  
FirstCurrent, YCurrent 247  
FirstDataLogger, YDataLogger 289  
FirstDigitalIO, YDigitalIO 363

FirstDisplay, YDisplay 411  
FirstDualPower, YDualPower 493  
FirstFiles, YFiles 522  
FirstGenericSensor, YGenericSensor 556  
FirstGyro, YGyro 606  
FirstHubPort, YHubPort 660  
FirstHumidity, YHumidity 690  
FirstLed, YLed 732  
FirstLightSensor, YLightSensor 764  
FirstMagnetometer, YMagnetometer 808  
FirstModule, YModule 860  
FirstNetwork, YNetwork 907  
FirstOsControl, YOsControl 966  
FirstPower, YPower 994  
FirstPressure, YPressure 1041  
FirstPwmOutput, YPwmOutput 1084  
FirstPwmPowerSource, YPwmPowerSource 1125  
FirstQt, YQt 1153  
FirstRealTimeClock, YRealTimeClock 1195  
FirstRefFrame, YRefFrame 1227  
FirstRelay, YRelay 1267  
FirstSensor, YSensor 1307  
FirstServo, YServo 1350  
FirstTemperature, YTemperature 1389  
FirstTilt, YTilt 1434  
FirstVoc, YVoc 1477  
FirstVoltage, YVoltage 1520  
FirstVSource, YVSource 1562  
FirstWakeUpMonitor, YWakeUpMonitor 1599  
FirstWakeUpSchedule, YWakeUpSchedule 1638  
FirstWatchdog, YWatchdog 1679  
FirstWireless, YWireless 1727  
forgetAllDataStreams, YDataLogger 291  
format\_fs, YFiles 526  
Formatted 322  
Frame 1224  
FreeAPI, YAPI 16  
functionCount, YModule 863  
functionId, YModule 864  
functionName, YModule 865  
Functions 11  
functionValue, YModule 866

## G

General 11  
GenericSensor 553  
get\_3DCalibrationHint, YRefFrame 1230  
get\_3DCalibrationLogMsg, YRefFrame 1231  
get\_3DCalibrationProgress, YRefFrame 1232  
get\_3DCalibrationStage, YRefFrame 1233  
get\_3DCalibrationStageProgress, YRefFrame 1234  
get\_adminPassword, YNetwork 910  
get\_advertisedValue, YAccelerometer 42  
get\_advertisedValue, YAnButton 87  
get\_advertisedValue, YCarbonDioxide 130  
get\_advertisedValue, YColorLed 171  
get\_advertisedValue, YCompass 206

get\_advertisedValue, YCurrent 250  
 get\_advertisedValue, YDataLogger 292  
 get\_advertisedValue, YDigitalIO 366  
 get\_advertisedValue, YDisplay 415  
 get\_advertisedValue, YDualPower 495  
 get\_advertisedValue, YFiles 527  
 get\_advertisedValue, YGenericSensor 559  
 get\_advertisedValue, YGyro 609  
 get\_advertisedValue, YHubPort 662  
 get\_advertisedValue, YHumidity 693  
 get\_advertisedValue, YLed 734  
 get\_advertisedValue, YLightSensor 768  
 get\_advertisedValue, YMagnetometer 811  
 get\_advertisedValue, YNetwork 911  
 get\_advertisedValue, YOsControl 968  
 get\_advertisedValue, YPower 997  
 get\_advertisedValue, YPressure 1044  
 get\_advertisedValue, YPwmOutput 1087  
 get\_advertisedValue, YPwmPowerSource 1127  
 get\_advertisedValue, YQt 1156  
 get\_advertisedValue, YRealTimeClock 1197  
 get\_advertisedValue, YRefFrame 1235  
 get\_advertisedValue, YRelay 1270  
 get\_advertisedValue, YSensor 1310  
 get\_advertisedValue, YServo 1352  
 get\_advertisedValue, YTemperature 1392  
 get\_advertisedValue, YTilt 1437  
 get\_advertisedValue, YVoc 1480  
 get\_advertisedValue, YVoltage 1523  
 get\_advertisedValue, YVSource 1564  
 get\_advertisedValue, YWakeUpMonitor 1601  
 get\_advertisedValue, YWakeUpSchedule 1640  
 get\_advertisedValue, YWatchdog 1682  
 get\_advertisedValue, YWireless 1730  
 get\_analogCalibration, YAnButton 88  
 get\_autoStart, YDataLogger 293  
 get\_autoStart, YWatchdog 1683  
 get\_averageValue, YDataRun 322  
 get\_averageValue, YDataStream 346  
 get\_averageValue, YMeasure 851  
 get\_baudRate, YHubPort 663  
 get\_beacon, YModule 867  
 get\_bearing, YRefFrame 1236  
 get\_bitDirection, YDigitalIO 367  
 get\_bitOpenDrain, YDigitalIO 368  
 get\_bitPolarity, YDigitalIO 369  
 get\_bitState, YDigitalIO 370  
 get\_blinking, YLed 735  
 get\_brightness, YDisplay 416  
 get\_calibratedValue, YAnButton 89  
 get\_calibrationMax, YAnButton 90  
 get\_calibrationMin, YAnButton 91  
 get\_callbackCredentials, YNetwork 912  
 get\_callbackEncoding, YNetwork 913  
 get\_callbackMaxDelay, YNetwork 914  
 get\_callbackMethod, YNetwork 915  
 get\_callbackMinDelay, YNetwork 916  
 get\_callbackUrl, YNetwork 917  
 get\_channel, YWireless 1731  
 get\_columnCount, YDataStream 347  
 get\_columnNames, YDataStream 348  
 get\_cosPhi, YPower 998  
 get\_countdown, YRelay 1271  
 get\_countdown, YWatchdog 1684  
 get\_currentRawValue, YAccelerometer 43  
 get\_currentRawValue, YCarbonDioxide 131  
 get\_currentRawValue, YCompass 207  
 get\_currentRawValue, YCurrent 251  
 get\_currentRawValue, YGenericSensor 560  
 get\_currentRawValue, YGyro 610  
 get\_currentRawValue, YHumidity 694  
 get\_currentRawValue, YLightSensor 769  
 get\_currentRawValue, YMagnetometer 812  
 get\_currentRawValue, YPower 999  
 get\_currentRawValue, YPressure 1045  
 get\_currentRawValue, YQt 1157  
 get\_currentRawValue, YSensor 1311  
 get\_currentRawValue, YTemperature 1393  
 get\_currentRawValue, YTilt 1438  
 get\_currentRawValue, YVoc 1481  
 get\_currentRawValue, YVoltage 1524  
 get\_currentRunIndex, YDataLogger 294  
 get\_currentValue, YAccelerometer 44  
 get\_currentValue, YCarbonDioxide 132  
 get\_currentValue, YCompass 208  
 get\_currentValue, YCurrent 252  
 get\_currentValue, YGenericSensor 561  
 get\_currentValue, YGyro 611  
 get\_currentValue, YHumidity 695  
 get\_currentValue, YLightSensor 770  
 get\_currentValue, YMagnetometer 813  
 get\_currentValue, YPower 1000  
 get\_currentValue, YPressure 1046  
 get\_currentValue, YQt 1158  
 get\_currentValue, YSensor 1312  
 get\_currentValue, YTemperature 1394  
 get\_currentValue, YTilt 1439  
 get\_currentValue, YVoc 1482  
 get\_currentValue, YVoltage 1525  
 get\_data, YDataStream 349  
 get\_dataRows, YDataStream 350  
 get\_dataSamplesIntervalMs, YDataStream 351  
 get\_dataSets, YDataLogger 295  
 get\_dataStreams, YDataLogger 296  
 get\_dateTime, YRealTimeClock 1198  
 get\_detectedWlans, YWireless 1732  
 get\_discoverable, YNetwork 918  
 get\_display, YDisplayLayer 471  
 get\_displayHeight, YDisplay 417  
 get\_displayHeight, YDisplayLayer 472  
 get\_displayLayer, YDisplay 418  
 get\_displayType, YDisplay 419  
 get\_displayWidth, YDisplay 420  
 get\_displayWidth, YDisplayLayer 473  
 get\_duration, YDataRun 323  
 get\_duration, YDataStream 352  
 get\_dutyCycle, YPwmOutput 1088  
 get\_dutyCycleAtPowerOn, YPwmOutput 1089

get\_enabled, YDisplay 421  
 get\_enabled, YHubPort 664  
 get\_enabled, YPwmOutput 1090  
 get\_enabled, YServo 1353  
 get\_enabledAtPowerOn, YPwmOutput 1091  
 get\_enabledAtPowerOn, YServo 1354  
 get\_endTimeUTC, YDataSet 333  
 get\_endTimeUTC, YMeasure 852  
 get\_errorMessage, YAccelerometer 45  
 get\_errorMessage, YAnButton 92  
 get\_errorMessage, YCarbonDioxide 133  
 get\_errorMessage, YColorLed 172  
 get\_errorMessage, YCompass 209  
 get\_errorMessage, YCurrent 253  
 get\_errorMessage, YDataLogger 297  
 get\_errorMessage, YDigitalIO 371  
 get\_errorMessage, YDisplay 422  
 get\_errorMessage, YDualPower 496  
 get\_errorMessage, YFiles 528  
 get\_errorMessage, YGenericSensor 562  
 get\_errorMessage, YGyro 612  
 get\_errorMessage, YHubPort 665  
 get\_errorMessage, YHumidity 696  
 get\_errorMessage, YLed 736  
 get\_errorMessage, YLightSensor 771  
 get\_errorMessage, YMagnetometer 814  
 get\_errorMessage, YModule 868  
 get\_errorMessage, YNetwork 919  
 get\_errorMessage, YOsControl 969  
 get\_errorMessage, YPower 1001  
 get\_errorMessage, YPressure 1047  
 get\_errorMessage, YPwmOutput 1092  
 get\_errorMessage, YPwmPowerSource 1128  
 get\_errorMessage, YQt 1159  
 get\_errorMessage, YRealTimeClock 1199  
 get\_errorMessage, YRefFrame 1237  
 get\_errorMessage, YRelay 1272  
 get\_errorMessage, YSensor 1313  
 get\_errorMessage, YServo 1355  
 get\_errorMessage, YTemperature 1395  
 get\_errorMessage, YTilt 1440  
 get\_errorMessage, YVoc 1483  
 get\_errorMessage, YVoltage 1526  
 get\_errorMessage, YVSource 1565  
 get\_errorMessage, YWakeUpMonitor 1602  
 get\_errorMessage, YWakeUpSchedule 1641  
 get\_errorMessage, YWatchdog 1685  
 get\_errorMessage, YWireless 1733  
 get\_errorType, YAccelerometer 46  
 get\_errorType, YAnButton 93  
 get\_errorType, YCarbonDioxide 134  
 get\_errorType, YColorLed 173  
 get\_errorType, YCompass 210  
 get\_errorType, YCurrent 254  
 get\_errorType, YDataLogger 298  
 get\_errorType, YDigitalIO 372  
 get\_errorType, YDisplay 423  
 get\_errorType, YDualPower 497  
 get\_errorType, YFiles 529  
 get\_errorType, YGenericSensor 563  
 get\_errorType, YGyro 613  
 get\_errorType, YHubPort 666  
 get\_errorType, YHumidity 697  
 get\_errorType, YLed 737  
 get\_errorType, YLightSensor 772  
 get\_errorType, YMagnetometer 815  
 get\_errorType, YModule 869  
 get\_errorType, YNetwork 920  
 get\_errorType, YOsControl 970  
 get\_errorType, YPower 1002  
 get\_errorType, YPressure 1048  
 get\_errorType, YPwmOutput 1093  
 get\_errorType, YPwmPowerSource 1129  
 get\_errorType, YQt 1160  
 get\_errorType, YRealTimeClock 1200  
 get\_errorType, YRefFrame 1238  
 get\_errorType, YRelay 1273  
 get\_errorType, YSensor 1314  
 get\_errorType, YServo 1356  
 get\_errorType, YTemperature 1396  
 get\_errorType, YTilt 1441  
 get\_errorType, YVoc 1484  
 get\_errorType, YVoltage 1527  
 get\_errorType, YVSource 1566  
 get\_errorType, YWakeUpMonitor 1603  
 get\_errorType, YWakeUpSchedule 1642  
 get\_errorType, YWatchdog 1686  
 get\_errorType, YWireless 1734  
 get\_extPowerFailure, YVSource 1567  
 get\_extVoltage, YDualPower 498  
 get\_failure, YVSource 1568  
 get\_filesCount, YFiles 530  
 get\_firmwareRelease, YModule 870  
 get\_freeSpace, YFiles 531  
 get\_frequency, YPwmOutput 1094  
 get\_friendlyName, YAccelerometer 47  
 get\_friendlyName, YAnButton 94  
 get\_friendlyName, YCarbonDioxide 135  
 get\_friendlyName, YColorLed 174  
 get\_friendlyName, YCompass 211  
 get\_friendlyName, YCurrent 255  
 get\_friendlyName, YDataLogger 299  
 get\_friendlyName, YDigitalIO 373  
 get\_friendlyName, YDisplay 424  
 get\_friendlyName, YDualPower 499  
 get\_friendlyName, YFiles 532  
 get\_friendlyName, YGenericSensor 564  
 get\_friendlyName, YGyro 614  
 get\_friendlyName, YHubPort 667  
 get\_friendlyName, YHumidity 698  
 get\_friendlyName, YLed 738  
 get\_friendlyName, YLightSensor 773  
 get\_friendlyName, YMagnetometer 816  
 get\_friendlyName, YNetwork 921  
 get\_friendlyName, YOsControl 971  
 get\_friendlyName, YPower 1003  
 get\_friendlyName, YPressure 1049  
 get\_friendlyName, YPwmOutput 1095

get\_friendlyName, YPwmPowerSource 1130  
get\_friendlyName, YQt 1161  
get\_friendlyName, YRealTimeClock 1201  
get\_friendlyName, YRefFrame 1239  
get\_friendlyName, YRelay 1274  
get\_friendlyName, YSensor 1315  
get\_friendlyName, YServo 1357  
get\_friendlyName, YTemperature 1397  
get\_friendlyName, YTilt 1442  
get\_friendlyName, YVoc 1485  
get\_friendlyName, YVoltage 1528  
get\_friendlyName, YVSource 1569  
get\_friendlyName, YWakeUpMonitor 1604  
get\_friendlyName, YWakeUpSchedule 1643  
get\_friendlyName, YWatchdog 1687  
get\_friendlyName, YWireless 1735  
get\_functionDescriptor, YAccelerometer 48  
get\_functionDescriptor, YAnButton 95  
get\_functionDescriptor, YCarbonDioxide 136  
get\_functionDescriptor, YColorLed 175  
get\_functionDescriptor, YCompass 212  
get\_functionDescriptor, YCurrent 256  
get\_functionDescriptor, YDataLogger 300  
get\_functionDescriptor, YDigitalIO 374  
get\_functionDescriptor, YDisplay 425  
get\_functionDescriptor, YDualPower 500  
get\_functionDescriptor, YFiles 533  
get\_functionDescriptor, YGenericSensor 565  
get\_functionDescriptor, YGyro 615  
get\_functionDescriptor, YHubPort 668  
get\_functionDescriptor, YHumidity 699  
get\_functionDescriptor, YLed 739  
get\_functionDescriptor, YLightSensor 774  
get\_functionDescriptor, YMagnetometer 817  
get\_functionDescriptor, YNetwork 922  
get\_functionDescriptor, YOsControl 972  
get\_functionDescriptor, YPower 1004  
get\_functionDescriptor, YPressure 1050  
get\_functionDescriptor, YPwmOutput 1096  
get\_functionDescriptor, YPwmPowerSource 1131  
get\_functionDescriptor, YQt 1162  
get\_functionDescriptor, YRealTimeClock 1202  
get\_functionDescriptor, YRefFrame 1240  
get\_functionDescriptor, YRelay 1275  
get\_functionDescriptor, YSensor 1316  
get\_functionDescriptor, YServo 1358  
get\_functionDescriptor, YTemperature 1398  
get\_functionDescriptor, YTilt 1443  
get\_functionDescriptor, YVoc 1486  
get\_functionDescriptor, YVoltage 1529  
get\_functionDescriptor, YVSource 1570  
get\_functionDescriptor, YWakeUpMonitor 1605  
get\_functionDescriptor, YWakeUpSchedule 1644  
get\_functionDescriptor, YWatchdog 1688  
get\_functionDescriptor, YWireless 1736  
get\_functionId, YAccelerometer 49  
get\_functionId, YAnButton 96  
get\_functionId, YCarbonDioxide 137  
get\_functionId, YColorLed 176

get\_functionId, YCompass 213  
get\_functionId, YCurrent 257  
get\_functionId, YDataLogger 301  
get\_functionId, YDataSet 334  
get\_functionId, YDigitalIO 375  
get\_functionId, YDisplay 426  
get\_functionId, YDualPower 501  
get\_functionId, YFiles 534  
get\_functionId, YGenericSensor 566  
get\_functionId, YGyro 616  
get\_functionId, YHubPort 669  
get\_functionId, YHumidity 700  
get\_functionId, YLed 740  
get\_functionId, YLightSensor 775  
get\_functionId, YMagnetometer 818  
get\_functionId, YNetwork 923  
get\_functionId, YOsControl 973  
get\_functionId, YPower 1005  
get\_functionId, YPressure 1051  
get\_functionId, YPwmOutput 1097  
get\_functionId, YPwmPowerSource 1132  
get\_functionId, YQt 1163  
get\_functionId, YRealTimeClock 1203  
get\_functionId, YRefFrame 1241  
get\_functionId, YRelay 1276  
get\_functionId, YSensor 1317  
get\_functionId, YServo 1359  
get\_functionId, YTemperature 1399  
get\_functionId, YTilt 1444  
get\_functionId, YVoc 1487  
get\_functionId, YVoltage 1530  
get\_functionId, YVSource 1571  
get\_functionId, YWakeUpMonitor 1606  
get\_functionId, YWakeUpSchedule 1645  
get\_functionId, YWatchdog 1689  
get\_functionId, YWireless 1737  
get\_hardwareId, YAccelerometer 50  
get\_hardwareId, YAnButton 97  
get\_hardwareId, YCarbonDioxide 138  
get\_hardwareId, YColorLed 177  
get\_hardwareId, YCompass 214  
get\_hardwareId, YCurrent 258  
get\_hardwareId, YDataLogger 302  
get\_hardwareId, YDataSet 335  
get\_hardwareId, YDigitalIO 376  
get\_hardwareId, YDisplay 427  
get\_hardwareId, YDualPower 502  
get\_hardwareId, YFiles 535  
get\_hardwareId, YGenericSensor 567  
get\_hardwareId, YGyro 617  
get\_hardwareId, YHubPort 670  
get\_hardwareId, YHumidity 701  
get\_hardwareId, YLed 741  
get\_hardwareId, YLightSensor 776  
get\_hardwareId, YMagnetometer 819  
get\_hardwareId, YModule 871  
get\_hardwareId, YNetwork 924  
get\_hardwareId, YOsControl 974  
get\_hardwareId, YPower 1006

get\_hardwareId, YPressure 1052  
get\_hardwareId, YPwmOutput 1098  
get\_hardwareId, YPwmPowerSource 1133  
get\_hardwareId, YQt 1164  
get\_hardwareId, YRealTimeClock 1204  
get\_hardwareId, YRefFrame 1242  
get\_hardwareId, YRelay 1277  
get\_hardwareId, YSensor 1318  
get\_hardwareId, YServo 1360  
get\_hardwareId, YTemperature 1400  
get\_hardwareId, YTilt 1445  
get\_hardwareId, YVoc 1488  
get\_hardwareId, YVoltage 1531  
get\_hardwareId, YVSource 1572  
get\_hardwareId, YWakeUpMonitor 1607  
get\_hardwareId, YWakeUpSchedule 1646  
get\_hardwareId, YWatchdog 1690  
get\_hardwareId, YWireless 1738  
get\_heading, YGyro 618  
get\_highestValue, YAccelerometer 51  
get\_highestValue, YCarbonDioxide 139  
get\_highestValue, YCompass 215  
get\_highestValue, YCurrent 259  
get\_highestValue, YGenericSensor 568  
get\_highestValue, YGyro 619  
get\_highestValue, YHumidity 702  
get\_highestValue, YLightSensor 777  
get\_highestValue, YMagnetometer 820  
get\_highestValue, YPower 1007  
get\_highestValue, YPressure 1053  
get\_highestValue, YQt 1165  
get\_highestValue, YSensor 1319  
get\_highestValue, YTemperature 1401  
get\_highestValue, YTilt 1446  
get\_highestValue, YVoc 1489  
get\_highestValue, YVoltage 1532  
get\_hours, YWakeUpSchedule 1647  
get\_hslColor, YColorLed 178  
get\_icon2d, YModule 872  
get\_ipAddress, YNetwork 925  
get\_isPressed, YAnButton 98  
get\_lastLogs, YModule 873  
get\_lastTimePressed, YAnButton 99  
get\_lastTimeReleased, YAnButton 100  
get\_layerCount, YDisplay 428  
get\_layerHeight, YDisplay 429  
get\_layerHeight, YDisplayLayer 474  
get\_layerWidth, YDisplay 430  
get\_layerWidth, YDisplayLayer 475  
get\_linkQuality, YWireless 1739  
get\_list, YFiles 536  
get\_logFrequency, YAccelerometer 52  
get\_logFrequency, YCarbonDioxide 140  
get\_logFrequency, YCompass 216  
get\_logFrequency, YCurrent 260  
get\_logFrequency, YGenericSensor 569  
get\_logFrequency, YGyro 620  
get\_logFrequency, YHumidity 703  
get\_logFrequency, YLightSensor 778

get\_logFrequency, YMagnetometer 821  
get\_logFrequency, YPower 1008  
get\_logFrequency, YPressure 1054  
get\_logFrequency, YQt 1166  
get\_logFrequency, YSensor 1320  
get\_logFrequency, YTemperature 1402  
get\_logFrequency, YTilt 1447  
get\_logFrequency, YVoc 1490  
get\_logFrequency, YVoltage 1533  
get\_logicalName, YAccelerometer 53  
get\_logicalName, YAnButton 101  
get\_logicalName, YCarbonDioxide 141  
get\_logicalName, YColorLed 179  
get\_logicalName, YCompass 217  
get\_logicalName, YCurrent 261  
get\_logicalName, YDataLogger 303  
get\_logicalName, YDigitalIO 377  
get\_logicalName, YDisplay 431  
get\_logicalName, YDualPower 503  
get\_logicalName, YFiles 537  
get\_logicalName, YGenericSensor 570  
get\_logicalName, YGyro 621  
get\_logicalName, YHubPort 671  
get\_logicalName, YHumidity 704  
get\_logicalName, YLed 742  
get\_logicalName, YLightSensor 779  
get\_logicalName, YMagnetometer 822  
get\_logicalName, YModule 874  
get\_logicalName, YNetwork 926  
get\_logicalName, YOsControl 975  
get\_logicalName, YPower 1009  
get\_logicalName, YPressure 1055  
get\_logicalName, YPwmOutput 1099  
get\_logicalName, YPwmPowerSource 1134  
get\_logicalName, YQt 1167  
get\_logicalName, YRealTimeClock 1205  
get\_logicalName, YRefFrame 1243  
get\_logicalName, YRelay 1278  
get\_logicalName, YSensor 1321  
get\_logicalName, YServo 1361  
get\_logicalName, YTemperature 1403  
get\_logicalName, YTilt 1448  
get\_logicalName, YVoc 1491  
get\_logicalName, YVoltage 1534  
get\_logicalName, YVSource 1573  
get\_logicalName, YWakeUpMonitor 1608  
get\_logicalName, YWakeUpSchedule 1648  
get\_logicalName, YWatchdog 1691  
get\_logicalName, YWireless 1740  
get\_lowestValue, YAccelerometer 54  
get\_lowestValue, YCarbonDioxide 142  
get\_lowestValue, YCompass 218  
get\_lowestValue, YCurrent 262  
get\_lowestValue, YGenericSensor 571  
get\_lowestValue, YGyro 622  
get\_lowestValue, YHumidity 705  
get\_lowestValue, YLightSensor 780  
get\_lowestValue, YMagnetometer 823  
get\_lowestValue, YPower 1010



get\_lowestValue, YPressure 1056  
get\_lowestValue, YQt 1168  
get\_lowestValue, YSensor 1322  
get\_lowestValue, YTemperature 1404  
get\_lowestValue, YTilt 1449  
get\_lowestValue, YVoc 1492  
get\_lowestValue, YVoltage 1535  
get\_luminosity, YLed 743  
get\_luminosity, YModule 875  
get\_macAddress, YNetwork 927  
get\_magneticHeading, YCompass 219  
get\_maxTimeOnStateA, YRelay 1279  
get\_maxTimeOnStateA, YWatchdog 1692  
get\_maxTimeOnStateB, YRelay 1280  
get\_maxTimeOnStateB, YWatchdog 1693  
get\_maxValue, YDataRun 324  
get\_maxValue, YDataStream 353  
get\_maxValue, YMeasure 853  
get\_measureNames, YDataRun 325  
get\_measures, YDataSet 336  
get\_message, YWireless 1741  
get\_meter, YPower 1011  
get\_meterTimer, YPower 1012  
get\_minutes, YWakeUpSchedule 1649  
get\_minutesA, YWakeUpSchedule 1650  
get\_minutesB, YWakeUpSchedule 1651  
get\_minValue, YDataRun 326  
get\_minValue, YDataStream 354  
get\_minValue, YMeasure 854  
get\_module, YAccelerometer 55  
get\_module, YAnButton 102  
get\_module, YCarbonDioxide 143  
get\_module, YColorLed 180  
get\_module, YCompass 220  
get\_module, YCurrent 263  
get\_module, YDataLogger 304  
get\_module, YDigitalIO 378  
get\_module, YDisplay 432  
get\_module, YDualPower 504  
get\_module, YFiles 538  
get\_module, YGenericSensor 572  
get\_module, YGyro 623  
get\_module, YHubPort 672  
get\_module, YHumidity 706  
get\_module, YLed 744  
get\_module, YLightSensor 781  
get\_module, YMagnetometer 824  
get\_module, YNetwork 928  
get\_module, YOsControl 976  
get\_module, YPower 1013  
get\_module, YPressure 1057  
get\_module, YPwmOutput 1100  
get\_module, YPwmPowerSource 1135  
get\_module, YQt 1169  
get\_module, YRealTimeClock 1206  
get\_module, YRefFrame 1244  
get\_module, YRelay 1281  
get\_module, YSensor 1323  
get\_module, YServo 1362

get\_module, YTemperature 1405  
get\_module, YTilt 1450  
get\_module, YVoc 1493  
get\_module, YVoltage 1536  
get\_module, YVSource 1574  
get\_module, YWakeUpMonitor 1609  
get\_module, YWakeUpSchedule 1652  
get\_module, YWatchdog 1694  
get\_module, YWireless 1742  
get\_module\_async, YAccelerometer 56  
get\_module\_async, YAnButton 103  
get\_module\_async, YCarbonDioxide 144  
get\_module\_async, YColorLed 181  
get\_module\_async, YCompass 221  
get\_module\_async, YCurrent 264  
get\_module\_async, YDataLogger 305  
get\_module\_async, YDigitalIO 379  
get\_module\_async, YDisplay 433  
get\_module\_async, YDualPower 505  
get\_module\_async, YFiles 539  
get\_module\_async, YGenericSensor 573  
get\_module\_async, YGyro 624  
get\_module\_async, YHubPort 673  
get\_module\_async, YHumidity 707  
get\_module\_async, YLed 745  
get\_module\_async, YLightSensor 782  
get\_module\_async, YMagnetometer 825  
get\_module\_async, YNetwork 929  
get\_module\_async, YOsControl 977  
get\_module\_async, YPower 1014  
get\_module\_async, YPressure 1058  
get\_module\_async, YPwmOutput 1101  
get\_module\_async, YPwmPowerSource 1136  
get\_module\_async, YQt 1170  
get\_module\_async, YRealTimeClock 1207  
get\_module\_async, YRefFrame 1245  
get\_module\_async, YRelay 1282  
get\_module\_async, YSensor 1324  
get\_module\_async, YServo 1363  
get\_module\_async, YTemperature 1406  
get\_module\_async, YTilt 1451  
get\_module\_async, YVoc 1494  
get\_module\_async, YVoltage 1537  
get\_module\_async, YVSource 1575  
get\_module\_async, YWakeUpMonitor 1610  
get\_module\_async, YWakeUpSchedule 1653  
get\_module\_async, YWatchdog 1695  
get\_module\_async, YWireless 1743  
get\_monthDays, YWakeUpSchedule 1654  
get\_months, YWakeUpSchedule 1655  
get\_mountOrientation, YRefFrame 1246  
get\_mountPosition, YRefFrame 1247  
get\_neutral, YServo 1364  
get\_nextOccurrence, YWakeUpSchedule 1656  
get\_nextWakeUp, YWakeUpMonitor 1611  
get\_orientation, YDisplay 434  
get\_output, YRelay 1283  
get\_output, YWatchdog 1696  
get\_outputVoltage, YDigitalIO 380

get_overCurrent, YVSource	1576	get_reportFrequency, YCarbonDioxide	146
get_overHeat, YVSource	1577	get_reportFrequency, YCompass	223
get_overLoad, YVSource	1578	get_reportFrequency, YCurrent	266
get_period, YPwmOutput	1102	get_reportFrequency, YGenericSensor	575
get_persistentSettings, YModule	876	get_reportFrequency, YGyro	631
get_pitch, YGyro	625	get_reportFrequency, YHumidity	709
get_poeCurrent, YNetwork	930	get_reportFrequency, YLightSensor	784
get_portDirection, YDigitalIO	381	get_reportFrequency, YMagnetometer	827
get_portOpenDrain, YDigitalIO	382	get_reportFrequency, YPower	1016
get_portPolarity, YDigitalIO	383	get_reportFrequency, YPressure	1060
get_portSize, YDigitalIO	384	get_reportFrequency, YQt	1172
get_portState, YDigitalIO	385	get_reportFrequency, YSensor	1326
get_portState, YHubPort	674	get_reportFrequency, YTemperature	1408
get_position, YServo	1365	get_reportFrequency, YTilt	1453
get_positionAtPowerOn, YServo	1366	get_reportFrequency, YVoc	1496
get_power, YLed	746	get_reportFrequency, YVoltage	1539
get_powerControl, YDualPower	506	get_resolution, YAccelerometer	59
get_powerDuration, YWakeUpMonitor	1612	get_resolution, YCarbonDioxide	147
get_powerMode, YPwmPowerSource	1137	get_resolution, YCompass	224
get_powerState, YDualPower	507	get_resolution, YCurrent	267
get_preview, YDataSet	337	get_resolution, YGenericSensor	576
get_primaryDNS, YNetwork	931	get_resolution, YGyro	632
get_productId, YModule	877	get_resolution, YHumidity	710
get_productName, YModule	878	get_resolution, YLightSensor	785
get_productRelease, YModule	879	get_resolution, YMagnetometer	828
get_progress, YDataSet	338	get_resolution, YPower	1017
get_pulseCounter, YAnButton	104	get_resolution, YPressure	1061
get_pulseDuration, YPwmOutput	1103	get_resolution, YQt	1173
get_pulseTimer, YAnButton	105	get_resolution, YSensor	1327
get_pulseTimer, YRelay	1284	get_resolution, YTemperature	1409
get_pulseTimer, YWatchdog	1697	get_resolution, YTilt	1454
get_quaternionW, YGyro	626	get_resolution, YVoc	1497
get_quaternionX, YGyro	627	get_resolution, YVoltage	1540
get_quaternionY, YGyro	628	get_rgbColor, YColorLed	182
get_quaternionZ, YGyro	629	get_rgbColorAtPowerOn, YColorLed	183
get_range, YServo	1367	get_roll, YGyro	633
get_rawValue, YAnButton	106	get_router, YNetwork	933
get_readiness, YNetwork	932	get_rowCount, YDataStream	355
get_rebootCountdown, YModule	880	get_runIndex, YDataStream	356
get_recordedData, YAccelerometer	57	get_running, YWatchdog	1698
get_recordedData, YCarbonDioxide	145	get_secondaryDNS, YNetwork	934
get_recordedData, YCompass	222	get_security, YWireless	1744
get_recordedData, YCurrent	265	get_sensitivity, YAnButton	107
get_recordedData, YGenericSensor	574	get_sensorType, YTemperature	1410
get_recordedData, YGyro	630	get_serialNumber, YModule	881
get_recordedData, YHumidity	708	get_shutdownCountdown, YOsControl	978
get_recordedData, YLightSensor	783	get_signalRange, YGenericSensor	577
get_recordedData, YMagnetometer	826	get_signalUnit, YGenericSensor	578
get_recordedData, YPower	1015	get_signalValue, YGenericSensor	579
get_recordedData, YPressure	1059	get_sleepCountdown, YWakeUpMonitor	1613
get_recordedData, YQt	1171	get_ssid, YWireless	1745
get_recordedData, YSensor	1325	get_startTime, YDataStream	357
get_recordedData, YTemperature	1407	get_startTimeUTC, YDataRun	327
get_recordedData, YTilt	1452	get_startTimeUTC, YDataSet	339
get_recordedData, YVoc	1495	get_startTimeUTC, YDataStream	358
get_recordedData, YVoltage	1538	get_startTimeUTC, YMeasure	855
get_recording, YDataLogger	306	get_startupSeq, YDisplay	435
get_regulationFailure, YVSource	1579	get_state, YRelay	1285
get_reportFrequency, YAccelerometer	58	get_state, YWatchdog	1699

- get\_stateAtPowerOn, YRelay 1286
- get\_stateAtPowerOn, YWatchdog 1700
- get\_subnetMask, YNetwork 935
- get\_summary, YDataSet 340
- get\_timeSet, YRealTimeClock 1208
- get\_timeUTC, YDataLogger 307
- get\_triggerDelay, YWatchdog 1701
- get\_triggerDuration, YWatchdog 1702
- get\_unit, YAccelerometer 60
- get\_unit, YCarbonDioxide 148
- get\_unit, YCompass 225
- get\_unit, YCurrent 268
- get\_unit, YDataSet 341
- get\_unit, YGenericSensor 580
- get\_unit, YGyro 634
- get\_unit, YHumidity 711
- get\_unit, YLightSensor 786
- get\_unit, YMagnetometer 829
- get\_unit, YPower 1018
- get\_unit, YPressure 1062
- get\_unit, YQt 1174
- get\_unit, YSensor 1328
- get\_unit, YTemperature 1411
- get\_unit, YTilt 1455
- get\_unit, YVoc 1498
- get\_unit, YVoltage 1541
- get\_unit, YVSource 1580
- get\_unixTime, YRealTimeClock 1209
- get\_upTime, YModule 882
- get\_usbBandwidth, YModule 883
- get\_usbCurrent, YModule 884
- get\_userData, YAccelerometer 61
- get\_userData, YAnButton 108
- get\_userData, YCarbonDioxide 149
- get\_userData, YColorLed 184
- get\_userData, YCompass 226
- get\_userData, YCurrent 269
- get\_userData, YDataLogger 308
- get\_userData, YDigitalIO 386
- get\_userData, YDisplay 436
- get\_userData, YDualPower 508
- get\_userData, YFiles 540
- get\_userData, YGenericSensor 581
- get\_userData, YGyro 635
- get\_userData, YHubPort 675
- get\_userData, YHumidity 712
- get\_userData, YLed 747
- get\_userData, YLightSensor 787
- get\_userData, YMagnetometer 830
- get\_userData, YModule 885
- get\_userData, YNetwork 936
- get\_userData, YOsControl 979
- get\_userData, YPower 1019
- get\_userData, YPressure 1063
- get\_userData, YPwmOutput 1104
- get\_userData, YPwmPowerSource 1138
- get\_userData, YQt 1175
- get\_userData, YRealTimeClock 1210
- get\_userData, YRefFrame 1248

- get\_userData, YRelay 1287
- get\_userData, YSensor 1329
- get\_userData, YServo 1368
- get\_userData, YTemperature 1412
- get\_userData, YTilt 1456
- get\_userData, YVoc 1499
- get\_userData, YVoltage 1542
- get\_userData, YVSource 1581
- get\_userData, YWakeUpMonitor 1614
- get\_userData, YWakeUpSchedule 1657
- get\_userData, YWatchdog 1703
- get\_userData, YWireless 1746
- get\_userPassword, YNetwork 937
- get\_utcOffset, YRealTimeClock 1211
- get\_valueCount, YDataRun 328
- get\_valueInterval, YDataRun 329
- get\_valueRange, YGenericSensor 582
- get\_voltage, YVSource 1582
- get\_wakeUpReason, YWakeUpMonitor 1615
- get\_wakeUpState, YWakeUpMonitor 1616
- get\_weekDays, YWakeUpSchedule 1658
- get\_wwwWatchdogDelay, YNetwork 938
- get\_xValue, YAccelerometer 62
- get\_xValue, YGyro 636
- get\_xValue, YMagnetometer 831
- get\_yValue, YAccelerometer 63
- get\_yValue, YGyro 637
- get\_yValue, YMagnetometer 832
- get\_zValue, YAccelerometer 64
- get\_zValue, YGyro 638
- get\_zValue, YMagnetometer 833
- GetAPIVersion, YAPI 17
- GetTickCount, YAPI 18
- Gyroscope 603

## H

- HandleEvents, YAPI 19
- hide, YDisplayLayer 476
- hslMove, YColorLed 185
- Humidity 687

## I

- InitAPI, YAPI 20
- Interface 36, 82, 124, 167, 200, 244, 287, 360, 408, 459, 491, 520, 553, 603, 658, 687, 730, 761, 805, 857, 903, 991, 1038, 1081, 1123, 1150, 1193, 1264, 1304, 1347, 1386, 1431, 1474, 1517, 1560, 1596, 1635, 1676, 1725
- Introduction 1
- isOnline, YAccelerometer 65
- isOnline, YAnButton 109
- isOnline, YCarbonDioxide 150
- isOnline, YColorLed 186
- isOnline, YCompass 227
- isOnline, YCurrent 270
- isOnline, YDataLogger 309
- isOnline, YDigitalIO 387
- isOnline, YDisplay 437

isOnline, YDualPower 509  
isOnline, YFiles 541  
isOnline, YGenericSensor 583  
isOnline, YGyro 639  
isOnline, YHubPort 676  
isOnline, YHumidity 713  
isOnline, YLed 748  
isOnline, YLightSensor 788  
isOnline, YMagnetometer 834  
isOnline, YModule 886  
isOnline, YNetwork 939  
isOnline, YOsControl 980  
isOnline, YPower 1020  
isOnline, YPressure 1064  
isOnline, YPwmOutput 1105  
isOnline, YPwmPowerSource 1139  
isOnline, YQt 1176  
isOnline, YRealTimeClock 1212  
isOnline, YRefFrame 1249  
isOnline, YRelay 1288  
isOnline, YSensor 1330  
isOnline, YServo 1369  
isOnline, YTemperature 1413  
isOnline, YTilt 1457  
isOnline, YVoc 1500  
isOnline, YVoltage 1543  
isOnline, YVSource 1583  
isOnline, YWakeUpMonitor 1617  
isOnline, YWakeUpSchedule 1659  
isOnline, YWatchdog 1704  
isOnline, YWireless 1747  
isOnline\_async, YAccelerometer 66  
isOnline\_async, YAnButton 110  
isOnline\_async, YCarbonDioxide 151  
isOnline\_async, YColorLed 187  
isOnline\_async, YCompass 228  
isOnline\_async, YCurrent 271  
isOnline\_async, YDataLogger 310  
isOnline\_async, YDigitalIO 388  
isOnline\_async, YDisplay 438  
isOnline\_async, YDualPower 510  
isOnline\_async, YFiles 542  
isOnline\_async, YGenericSensor 584  
isOnline\_async, YGyro 640  
isOnline\_async, YHubPort 677  
isOnline\_async, YHumidity 714  
isOnline\_async, YLed 749  
isOnline\_async, YLightSensor 789  
isOnline\_async, YMagnetometer 835  
isOnline\_async, YModule 887  
isOnline\_async, YNetwork 940  
isOnline\_async, YOsControl 981  
isOnline\_async, YPower 1021  
isOnline\_async, YPressure 1065  
isOnline\_async, YPwmOutput 1106  
isOnline\_async, YPwmPowerSource 1140  
isOnline\_async, YQt 1177  
isOnline\_async, YRealTimeClock 1213  
isOnline\_async, YRefFrame 1250

isOnline\_async, YRelay 1289  
isOnline\_async, YSensor 1331  
isOnline\_async, YServo 1370  
isOnline\_async, YTemperature 1414  
isOnline\_async, YTilt 1458  
isOnline\_async, YVoc 1501  
isOnline\_async, YVoltage 1544  
isOnline\_async, YVSource 1584  
isOnline\_async, YWakeUpMonitor 1618  
isOnline\_async, YWakeUpSchedule 1660  
isOnline\_async, YWatchdog 1705  
isOnline\_async, YWireless 1748

## J

joinNetwork, YWireless 1749

## L

Library 3  
LightSensor 761  
lineTo, YDisplayLayer 477  
load, YAccelerometer 67  
load, YAnButton 111  
load, YCarbonDioxide 152  
load, YColorLed 188  
load, YCompass 229  
load, YCurrent 272  
load, YDataLogger 311  
load, YDigitalIO 389  
load, YDisplay 439  
load, YDualPower 511  
load, YFiles 543  
load, YGenericSensor 585  
load, YGyro 641  
load, YHubPort 678  
load, YHumidity 715  
load, YLed 750  
load, YLightSensor 790  
load, YMagnetometer 836  
load, YModule 888  
load, YNetwork 941  
load, YOsControl 982  
load, YPower 1022  
load, YPressure 1066  
load, YPwmOutput 1107  
load, YPwmPowerSource 1141  
load, YQt 1178  
load, YRealTimeClock 1214  
load, YRefFrame 1251  
load, YRelay 1290  
load, YSensor 1332  
load, YServo 1371  
load, YTemperature 1415  
load, YTilt 1459  
load, YVoc 1502  
load, YVoltage 1545  
load, YVSource 1585  
load, YWakeUpMonitor 1619  
load, YWakeUpSchedule 1661

- load, YWatchdog 1706
- load, YWireless 1750
- load\_async, YAccelerometer 69
- load\_async, YAnButton 112
- load\_async, YCarbonDioxide 154
- load\_async, YColorLed 189
- load\_async, YCompass 231
- load\_async, YCurrent 274
- load\_async, YDataLogger 312
- load\_async, YDigitalIO 390
- load\_async, YDisplay 440
- load\_async, YDualPower 512
- load\_async, YFiles 544
- load\_async, YGenericSensor 587
- load\_async, YGyro 643
- load\_async, YHubPort 679
- load\_async, YHumidity 717
- load\_async, YLed 751
- load\_async, YLightSensor 792
- load\_async, YMagnetometer 838
- load\_async, YModule 889
- load\_async, YNetwork 942
- load\_async, YOsControl 983
- load\_async, YPower 1024
- load\_async, YPressure 1068
- load\_async, YPwmOutput 1108
- load\_async, YPwmPowerSource 1142
- load\_async, YQt 1180
- load\_async, YRealTimeClock 1215
- load\_async, YRefFrame 1252
- load\_async, YRelay 1291
- load\_async, YSensor 1334
- load\_async, YServo 1372
- load\_async, YTemperature 1417
- load\_async, YTilt 1461
- load\_async, YVoc 1504
- load\_async, YVoltage 1547
- load\_async, YVSource 1586
- load\_async, YWakeUpMonitor 1620
- load\_async, YWakeUpSchedule 1662
- load\_async, YWatchdog 1707
- load\_async, YWireless 1751
- loadCalibrationPoints, YAccelerometer 68
- loadCalibrationPoints, YCarbonDioxide 153
- loadCalibrationPoints, YCompass 230
- loadCalibrationPoints, YCurrent 273
- loadCalibrationPoints, YGenericSensor 586
- loadCalibrationPoints, YGyro 642
- loadCalibrationPoints, YHumidity 716
- loadCalibrationPoints, YLightSensor 791
- loadCalibrationPoints, YMagnetometer 837
- loadCalibrationPoints, YPower 1023
- loadCalibrationPoints, YPressure 1067
- loadCalibrationPoints, YQt 1179
- loadCalibrationPoints, YSensor 1333
- loadCalibrationPoints, YTemperature 1416
- loadCalibrationPoints, YTilt 1460
- loadCalibrationPoints, YVoc 1503
- loadCalibrationPoints, YVoltage 1546

- loadMore, YDataSet 342
- loadMore\_async, YDataSet 343

## M

- Magnetometer 805
- Measured 851
- Module 5, 857
- more3DCalibration, YRefFrame 1253
- move, YServo 1373
- moveTo, YDisplayLayer 478

## N

- Network 903
- newSequence, YDisplay 441
- nextAccelerometer, YAccelerometer 70
- nextAnButton, YAnButton 113
- nextCarbonDioxide, YCarbonDioxide 155
- nextColorLed, YColorLed 190
- nextCompass, YCompass 232
- nextCurrent, YCurrent 275
- nextDataLogger, YDataLogger 313
- nextDigitalIO, YDigitalIO 391
- nextDisplay, YDisplay 442
- nextDualPower, YDualPower 513
- nextFiles, YFiles 545
- nextGenericSensor, YGenericSensor 588
- nextGyro, YGyro 644
- nextHubPort, YHubPort 680
- nextHumidity, YHumidity 718
- nextLed, YLed 752
- nextLightSensor, YLightSensor 793
- nextMagnetometer, YMagnetometer 839
- nextModule, YModule 890
- nextNetwork, YNetwork 943
- nextOsControl, YOsControl 984
- nextPower, YPower 1025
- nextPressure, YPressure 1069
- nextPwmOutput, YPwmOutput 1109
- nextPwmPowerSource, YPwmPowerSource 1143
- nextQt, YQt 1181
- nextRealTimeClock, YRealTimeClock 1216
- nextRefFrame, YRefFrame 1254
- nextRelay, YRelay 1292
- nextSensor, YSensor 1335
- nextServo, YServo 1374
- nextTemperature, YTemperature 1418
- nextTilt, YTilt 1462
- nextVoc, YVoc 1505
- nextVoltage, YVoltage 1548
- nextVSource, YVSource 1587
- nextWakeUpMonitor, YWakeUpMonitor 1621
- nextWakeUpSchedule, YWakeUpSchedule 1663
- nextWatchdog, YWatchdog 1708
- nextWireless, YWireless 1752

## O

Object 459

## P

pauseSequence, YDisplay 443  
ping, YNetwork 944  
playSequence, YDisplay 444  
Port 658  
Power 491, 991  
PreregisterHub, YAPI 21  
Pressure 1038  
pulse, YDigitalIO 392  
pulse, YRelay 1293  
pulse, YVSource 1588  
pulse, YWatchdog 1709  
pulseDurationMove, YPwmOutput 1110  
PwmPowerSource 1123  
Python 3

## Q

Quaternion 1150

## R

Real 1193  
reboot, YModule 891  
Recorded 332  
Reference 10, 1224  
registerAnglesCallback, YGyro 645  
RegisterDeviceArrivalCallback, YAPI 22  
RegisterDeviceRemovalCallback, YAPI 23  
RegisterHub, YAPI 24  
RegisterHubDiscoveryCallback, YAPI 25  
registerLogCallback, YModule 892  
RegisterLogFunction, YAPI 26  
registerQuaternionCallback, YGyro 646  
registerTimedReportCallback, YAccelerometer 71  
registerTimedReportCallback, YCarbonDioxide 156  
registerTimedReportCallback, YCompass 233  
registerTimedReportCallback, YCurrent 276  
registerTimedReportCallback, YGenericSensor 589  
registerTimedReportCallback, YGyro 647  
registerTimedReportCallback, YHumidity 719  
registerTimedReportCallback, YLightSensor 794  
registerTimedReportCallback, YMagnetometer 840  
registerTimedReportCallback, YPower 1026  
registerTimedReportCallback, YPressure 1070  
registerTimedReportCallback, YQt 1182  
registerTimedReportCallback, YSensor 1336  
registerTimedReportCallback, YTemperature 1419  
registerTimedReportCallback, YTilt 1463  
registerTimedReportCallback, YVoc 1506  
registerTimedReportCallback, YVoltage 1549

registerValueCallback, YAccelerometer 72  
registerValueCallback, YAnButton 114  
registerValueCallback, YCarbonDioxide 157  
registerValueCallback, YColorLed 191  
registerValueCallback, YCompass 234  
registerValueCallback, YCurrent 277  
registerValueCallback, YDataLogger 314  
registerValueCallback, YDigitalIO 393  
registerValueCallback, YDisplay 445  
registerValueCallback, YDualPower 514  
registerValueCallback, YFiles 546  
registerValueCallback, YGenericSensor 590  
registerValueCallback, YGyro 648  
registerValueCallback, YHubPort 681  
registerValueCallback, YHumidity 720  
registerValueCallback, YLed 753  
registerValueCallback, YLightSensor 795  
registerValueCallback, YMagnetometer 841  
registerValueCallback, YNetwork 945  
registerValueCallback, YOsControl 985  
registerValueCallback, YPower 1027  
registerValueCallback, YPressure 1071  
registerValueCallback, YPwmOutput 1111  
registerValueCallback, YPwmPowerSource 1144  
registerValueCallback, YQt 1183  
registerValueCallback, YRealTimeClock 1217  
registerValueCallback, YRefFrame 1255  
registerValueCallback, YRelay 1294  
registerValueCallback, YSensor 1337  
registerValueCallback, YServo 1375  
registerValueCallback, YTemperature 1420  
registerValueCallback, YTilt 1464  
registerValueCallback, YVoc 1507  
registerValueCallback, YVoltage 1550  
registerValueCallback, YVSource 1589  
registerValueCallback, YWakeUpMonitor 1622  
registerValueCallback, YWakeUpSchedule 1664  
registerValueCallback, YWatchdog 1710  
registerValueCallback, YWireless 1753  
Relay 1264  
remove, YFiles 547  
reset, YDisplayLayer 479  
reset, YPower 1028  
resetAll, YDisplay 446  
resetCounter, YAnButton 115  
resetSleepCountDown, YWakeUpMonitor 1623  
resetWatchdog, YWatchdog 1711  
revertFromFlash, YModule 893  
rgbMove, YColorLed 192

## S

save3DCalibration, YRefFrame 1256  
saveSequence, YDisplay 447  
saveToFlash, YModule 894  
SelectArchitecture, YAPI 27  
selectColorPen, YDisplayLayer 480  
selectEraser, YDisplayLayer 481  
selectFont, YDisplayLayer 482  
selectGrayPen, YDisplayLayer 483

Sensor 1304  
Sequence 322, 332, 345  
Servo 1347  
set\_adminPassword, YNetwork 946  
set\_analogCalibration, YAnButton 116  
set\_autoStart, YDataLogger 315  
set\_autoStart, YWatchdog 1712  
set\_beacon, YModule 895  
set\_bearing, YRefFrame 1257  
set\_bitDirection, YDigitalIO 394  
set\_bitOpenDrain, YDigitalIO 395  
set\_bitPolarity, YDigitalIO 396  
set\_bitState, YDigitalIO 397  
set\_blinking, YLed 754  
set\_brightness, YDisplay 448  
set\_calibrationMax, YAnButton 117  
set\_calibrationMin, YAnButton 118  
set\_callbackCredentials, YNetwork 947  
set\_callbackEncoding, YNetwork 948  
set\_callbackMaxDelay, YNetwork 949  
set\_callbackMethod, YNetwork 950  
set\_callbackMinDelay, YNetwork 951  
set\_callbackUrl, YNetwork 952  
set\_discoverable, YNetwork 953  
set\_dutyCycle, YPwmOutput 1112  
set\_dutyCycleAtPowerOn, YPwmOutput 1113  
set\_enabled, YDisplay 449  
set\_enabled, YHubPort 682  
set\_enabled, YPwmOutput 1114  
set\_enabled, YServo 1376  
set\_enabledAtPowerOn, YPwmOutput 1115  
set\_enabledAtPowerOn, YServo 1377  
set\_frequency, YPwmOutput 1116  
set\_highestValue, YAccelerometer 73  
set\_highestValue, YCarbonDioxide 158  
set\_highestValue, YCompass 235  
set\_highestValue, YCurrent 278  
set\_highestValue, YGenericSensor 591  
set\_highestValue, YGyro 649  
set\_highestValue, YHumidity 721  
set\_highestValue, YLightSensor 796  
set\_highestValue, YMagnetometer 842  
set\_highestValue, YPower 1029  
set\_highestValue, YPressure 1072  
set\_highestValue, YQt 1184  
set\_highestValue, YSensor 1338  
set\_highestValue, YTemperature 1421  
set\_highestValue, YTilt 1465  
set\_highestValue, YVoc 1508  
set\_highestValue, YVoltage 1551  
set\_hours, YWakeUpSchedule 1665  
set\_hslColor, YColorLed 193  
set\_logFrequency, YAccelerometer 74  
set\_logFrequency, YCarbonDioxide 159  
set\_logFrequency, YCompass 236  
set\_logFrequency, YCurrent 279  
set\_logFrequency, YGenericSensor 592  
set\_logFrequency, YGyro 650  
set\_logFrequency, YHumidity 722

set\_logFrequency, YLightSensor 797  
set\_logFrequency, YMagnetometer 843  
set\_logFrequency, YPower 1030  
set\_logFrequency, YPressure 1073  
set\_logFrequency, YQt 1185  
set\_logFrequency, YSensor 1339  
set\_logFrequency, YTemperature 1422  
set\_logFrequency, YTilt 1466  
set\_logFrequency, YVoc 1509  
set\_logFrequency, YVoltage 1552  
set\_logicalName, YAccelerometer 75  
set\_logicalName, YAnButton 119  
set\_logicalName, YCarbonDioxide 160  
set\_logicalName, YColorLed 194  
set\_logicalName, YCompass 237  
set\_logicalName, YCurrent 280  
set\_logicalName, YDataLogger 316  
set\_logicalName, YDigitalIO 398  
set\_logicalName, YDisplay 450  
set\_logicalName, YDualPower 515  
set\_logicalName, YFiles 548  
set\_logicalName, YGenericSensor 593  
set\_logicalName, YGyro 651  
set\_logicalName, YHubPort 683  
set\_logicalName, YHumidity 723  
set\_logicalName, YLed 755  
set\_logicalName, YLightSensor 798  
set\_logicalName, YMagnetometer 844  
set\_logicalName, YModule 896  
set\_logicalName, YNetwork 954  
set\_logicalName, YOsControl 986  
set\_logicalName, YPower 1031  
set\_logicalName, YPressure 1074  
set\_logicalName, YPwmOutput 1117  
set\_logicalName, YPwmPowerSource 1145  
set\_logicalName, YQt 1186  
set\_logicalName, YRealTimeClock 1218  
set\_logicalName, YRefFrame 1258  
set\_logicalName, YRelay 1295  
set\_logicalName, YSensor 1340  
set\_logicalName, YServo 1378  
set\_logicalName, YTemperature 1423  
set\_logicalName, YTilt 1467  
set\_logicalName, YVoc 1510  
set\_logicalName, YVoltage 1553  
set\_logicalName, YVSource 1590  
set\_logicalName, YWakeUpMonitor 1624  
set\_logicalName, YWakeUpSchedule 1666  
set\_logicalName, YWatchdog 1713  
set\_logicalName, YWireless 1754  
set\_lowestValue, YAccelerometer 76  
set\_lowestValue, YCarbonDioxide 161  
set\_lowestValue, YCompass 238  
set\_lowestValue, YCurrent 281  
set\_lowestValue, YGenericSensor 594  
set\_lowestValue, YGyro 652  
set\_lowestValue, YHumidity 724  
set\_lowestValue, YLightSensor 799  
set\_lowestValue, YMagnetometer 845

set\_lowestValue, YPower 1032  
 set\_lowestValue, YPressure 1075  
 set\_lowestValue, YQt 1187  
 set\_lowestValue, YSensor 1341  
 set\_lowestValue, YTemperature 1424  
 set\_lowestValue, YTilt 1468  
 set\_lowestValue, YVoc 1511  
 set\_lowestValue, YVoltage 1554  
 set\_luminosity, YLed 756  
 set\_luminosity, YModule 897  
 set\_maxTimeOnStateA, YRelay 1296  
 set\_maxTimeOnStateA, YWatchdog 1714  
 set\_maxTimeOnStateB, YRelay 1297  
 set\_maxTimeOnStateB, YWatchdog 1715  
 set\_minutes, YWakeUpSchedule 1667  
 set\_minutesA, YWakeUpSchedule 1668  
 set\_minutesB, YWakeUpSchedule 1669  
 set\_monthDays, YWakeUpSchedule 1670  
 set\_months, YWakeUpSchedule 1671  
 set\_mountPosition, YRefFrame 1259  
 set\_neutral, YServo 1379  
 set\_nextWakeUp, YWakeUpMonitor 1625  
 set\_orientation, YDisplay 451  
 set\_output, YRelay 1298  
 set\_output, YWatchdog 1716  
 set\_outputVoltage, YDigitalIO 399  
 set\_period, YPwmOutput 1118  
 set\_portDirection, YDigitalIO 400  
 set\_portOpenDrain, YDigitalIO 401  
 set\_portPolarity, YDigitalIO 402  
 set\_portState, YDigitalIO 403  
 set\_position, YServo 1380  
 set\_positionAtPowerOn, YServo 1381  
 set\_power, YLed 757  
 set\_powerControl, YDualPower 516  
 set\_powerDuration, YWakeUpMonitor 1626  
 set\_powerMode, YPwmPowerSource 1146  
 set\_primaryDNS, YNetwork 955  
 set\_pulseDuration, YPwmOutput 1119  
 set\_range, YServo 1382  
 set\_recording, YDataLogger 317  
 set\_reportFrequency, YAccelerometer 77  
 set\_reportFrequency, YCarbonDioxide 162  
 set\_reportFrequency, YCompass 239  
 set\_reportFrequency, YCurrent 282  
 set\_reportFrequency, YGenericSensor 595  
 set\_reportFrequency, YGyro 653  
 set\_reportFrequency, YHumidity 725  
 set\_reportFrequency, YLightSensor 800  
 set\_reportFrequency, YMagnetometer 846  
 set\_reportFrequency, YPower 1033  
 set\_reportFrequency, YPressure 1076  
 set\_reportFrequency, YQt 1188  
 set\_reportFrequency, YSensor 1342  
 set\_reportFrequency, YTemperature 1425  
 set\_reportFrequency, YTilt 1469  
 set\_reportFrequency, YVoc 1512  
 set\_reportFrequency, YVoltage 1555  
 set\_resolution, YAccelerometer 78  
 set\_resolution, YCarbonDioxide 163  
 set\_resolution, YCompass 240  
 set\_resolution, YCurrent 283  
 set\_resolution, YGenericSensor 596  
 set\_resolution, YGyro 654  
 set\_resolution, YHumidity 726  
 set\_resolution, YLightSensor 801  
 set\_resolution, YMagnetometer 847  
 set\_resolution, YPower 1034  
 set\_resolution, YPressure 1077  
 set\_resolution, YQt 1189  
 set\_resolution, YSensor 1343  
 set\_resolution, YTemperature 1426  
 set\_resolution, YTilt 1470  
 set\_resolution, YVoc 1513  
 set\_resolution, YVoltage 1556  
 set\_rgbColor, YColorLed 195  
 set\_rgbColorAtPowerOn, YColorLed 196  
 set\_running, YWatchdog 1717  
 set\_secondaryDNS, YNetwork 956  
 set\_sensitivity, YAnButton 120  
 set\_sensorType, YTemperature 1427  
 set\_signalRange, YGenericSensor 597  
 set\_sleepCountdown, YWakeUpMonitor 1627  
 set\_startupSeq, YDisplay 452  
 set\_state, YRelay 1299  
 set\_state, YWatchdog 1718  
 set\_stateAtPowerOn, YRelay 1300  
 set\_stateAtPowerOn, YWatchdog 1719  
 set\_timeUTC, YDataLogger 318  
 set\_triggerDelay, YWatchdog 1720  
 set\_triggerDuration, YWatchdog 1721  
 set\_unit, YGenericSensor 598  
 set\_unixTime, YRealTimeClock 1219  
 set\_usbBandwidth, YModule 898  
 set\_userData, YAccelerometer 79  
 set\_userData, YAnButton 121  
 set\_userData, YCarbonDioxide 164  
 set\_userData, YColorLed 197  
 set\_userData, YCompass 241  
 set\_userData, YCurrent 284  
 set\_userData, YDataLogger 319  
 set\_userData, YDigitalIO 404  
 set\_userData, YDisplay 453  
 set\_userData, YDualPower 517  
 set\_userData, YFiles 549  
 set\_userData, YGenericSensor 599  
 set\_userData, YGyro 655  
 set\_userData, YHubPort 684  
 set\_userData, YHumidity 727  
 set\_userData, YLed 758  
 set\_userData, YLightSensor 802  
 set\_userData, YMagnetometer 848  
 set\_userData, YModule 899  
 set\_userData, YNetwork 957  
 set\_userData, YOsControl 987  
 set\_userData, YPower 1035  
 set\_userData, YPressure 1078  
 set\_userData, YPwmOutput 1120



set\_userdata, YPwmPowerSource 1147  
set\_userdata, YQt 1190  
set\_userdata, YRealTimeClock 1220  
set\_userdata, YRefFrame 1260  
set\_userdata, YRelay 1301  
set\_userdata, YSensor 1344  
set\_userdata, YServo 1383  
set\_userdata, YTemperature 1428  
set\_userdata, YTilt 1471  
set\_userdata, YVoc 1514  
set\_userdata, YVoltage 1557  
set\_userdata, YVSource 1591  
set\_userdata, YWakeUpMonitor 1628  
set\_userdata, YWakeUpSchedule 1672  
set\_userdata, YWatchdog 1722  
set\_userdata, YWireless 1755  
set\_userPassword, YNetwork 958  
set\_utcOffset, YRealTimeClock 1221  
set\_valueInterval, YDataRun 330  
set\_valueRange, YGenericSensor 600  
set\_voltage, YVSource 1592  
set\_weekDays, YWakeUpSchedule 1673  
set\_wwwWatchdogDelay, YNetwork 959  
setAntialiasingMode, YDisplayLayer 484  
setConsoleBackground, YDisplayLayer 485  
setConsoleMargins, YDisplayLayer 486  
setConsoleWordWrap, YDisplayLayer 487  
SetDelegate, YAPI 28  
setLayerPosition, YDisplayLayer 488  
SetTimeout, YAPI 29  
shutdown, YOsControl 988  
Sleep, YAPI 30  
sleep, YWakeUpMonitor 1629  
sleepFor, YWakeUpMonitor 1630  
sleepUntil, YWakeUpMonitor 1631  
Source 3, 1560  
start3DCalibration, YRefFrame 1261  
stopSequence, YDisplay 454  
Supply 491  
swapLayerContent, YDisplay 455

## T

Temperature 1386  
Tilt 1431  
Time 1193  
toggle\_bitState, YDigitalIO 405  
triggerFirmwareUpdate, YModule 900  
TriggerHubDiscovery, YAPI 31

## U

Unformatted 345  
unhide, YDisplayLayer 489  
UnregisterHub, YAPI 32  
UpdateDeviceList, YAPI 33  
UpdateDeviceList\_async, YAPI 34  
upload, YDisplay 456  
upload, YFiles 550  
useDHCP, YNetwork 960

useStaticIP, YNetwork 961

## V

Value 851  
Voltage 1517, 1560  
voltageMove, YVSource 1593

## W

wait\_async, YAccelerometer 80  
wait\_async, YAnButton 122  
wait\_async, YCarbonDioxide 165  
wait\_async, YColorLed 198  
wait\_async, YCompass 242  
wait\_async, YCurrent 285  
wait\_async, YDataLogger 320  
wait\_async, YDigitalIO 406  
wait\_async, YDisplay 457  
wait\_async, YDualPower 518  
wait\_async, YFiles 551  
wait\_async, YGenericSensor 601  
wait\_async, YGyro 656  
wait\_async, YHubPort 685  
wait\_async, YHumidity 728  
wait\_async, YLed 759  
wait\_async, YLightSensor 803  
wait\_async, YMagnetometer 849  
wait\_async, YModule 901  
wait\_async, YNetwork 962  
wait\_async, YOsControl 989  
wait\_async, YPower 1036  
wait\_async, YPressure 1079  
wait\_async, YPwmOutput 1121  
wait\_async, YPwmPowerSource 1148  
wait\_async, YQt 1191  
wait\_async, YRealTimeClock 1222  
wait\_async, YRefFrame 1262  
wait\_async, YRelay 1302  
wait\_async, YSensor 1345  
wait\_async, YServo 1384  
wait\_async, YTemperature 1429  
wait\_async, YTilt 1472  
wait\_async, YVoc 1515  
wait\_async, YVoltage 1558  
wait\_async, YVSource 1594  
wait\_async, YWakeUpMonitor 1632  
wait\_async, YWakeUpSchedule 1674  
wait\_async, YWatchdog 1723  
wait\_async, YWireless 1756  
wakeUp, YWakeUpMonitor 1633  
WakeUpMonitor 1596  
WakeUpSchedule 1635  
Watchdog 1676  
Wireless 1725

## Y

YAccelerometer 38-80  
YAnButton 84-122

- YAPI 12-34
- YCarbonDioxide 126-165
- yCheckLogicalName 12
- YColorLed 168-198
- YCompass 202-242
- YCurrent 246-285
- YDataLogger 288-320
- YDataRun 322-330
- YDataSet 333-343
- YDataStream 346-358
- YDigitalIO 362-406
- yDisableExceptions 13
- YDisplay 410-457
- YDisplayLayer 460-489
- YDualPower 492-518
- yEnableExceptions 14
- yEnableUSBHost 15
- YFiles 521-551
- yFindAccelerometer 38
- yFindAnButton 84
- yFindCarbonDioxide 126
- yFindColorLed 168
- yFindCompass 202
- yFindCurrent 246
- yFindDataLogger 288
- yFindDigitalIO 362
- yFindDisplay 410
- yFindDualPower 492
- yFindFiles 521
- yFindGenericSensor 555
- yFindGyro 605
- yFindHubPort 659
- yFindHumidity 689
- yFindLed 731
- yFindLightSensor 763
- yFindMagnetometer 807
- yFindModule 859
- yFindNetwork 906
- yFindOsControl 965
- yFindPower 993
- yFindPressure 1040
- yFindPwmOutput 1083
- yFindPwmPowerSource 1124
- yFindQt 1152
- yFindRealTimeClock 1194
- yFindRefFrame 1226
- yFindRelay 1266
- yFindSensor 1306
- yFindServo 1349
- yFindTemperature 1388
- yFindTilt 1433
- yFindVoc 1476
- yFindVoltage 1519
- yFindVSource 1561
- yFindWakeUpMonitor 1598
- yFindWakeUpSchedule 1637
- yFindWatchdog 1678
- yFindWireless 1726
- yFirstAccelerometer 39
- yFirstAnButton 85
- yFirstCarbonDioxide 127
- yFirstColorLed 169
- yFirstCompass 203
- yFirstCurrent 247
- yFirstDataLogger 289
- yFirstDigitalIO 363
- yFirstDisplay 411
- yFirstDualPower 493
- yFirstFiles 522
- yFirstGenericSensor 556
- yFirstGyro 606
- yFirstHubPort 660
- yFirstHumidity 690
- yFirstLed 732
- yFirstLightSensor 764
- yFirstMagnetometer 808
- yFirstModule 860
- yFirstNetwork 907
- yFirstOsControl 966
- yFirstPower 994
- yFirstPressure 1041
- yFirstPwmOutput 1084
- yFirstPwmPowerSource 1125
- yFirstQt 1153
- yFirstRealTimeClock 1195
- yFirstRefFrame 1227
- yFirstRelay 1267
- yFirstSensor 1307
- yFirstServo 1350
- yFirstTemperature 1389
- yFirstTilt 1434
- yFirstVoc 1477
- yFirstVoltage 1520
- yFirstVSource 1562
- yFirstWakeUpMonitor 1599
- yFirstWakeUpSchedule 1638
- yFirstWatchdog 1679
- yFirstWireless 1727
- yFreeAPI 16
- YGenericSensor 555-601
- yGetAPIVersion 17
- yGetTickCount 18
- YGyro 605-656
- yHandleEvents 19
- YHubPort 659-685
- YHumidity 689-728
- yInitAPI 20
- YLed 731-759
- YLightSensor 763-803
- YMagnetometer 807-849
- YMeasure 851-855
- YModule 859-901
- YNetwork 906-962
- Yocto-Demo 3
- Yocto-hub 658
- YOsControl 965-989
- YPower 993-1036
- yPreregisterHub 21

YPressure	1040-1079	ySetDelegate	28
YPwmOutput	1083-1121	ySetTimeout	29
YPwmPowerSource	1124-1148	ySleep	30
YQt	1152-1191	YTemperature	1388-1429
YRealTimeClock	1194-1222	YTilt	1433-1472
YRefFrame	1226-1262	yTriggerHubDiscovery	31
yRegisterDeviceArrivalCallback	22	yUnregisterHub	32
yRegisterDeviceRemovalCallback	23	yUpdateDeviceList	33
yRegisterHub	24	yUpdateDeviceList_async	34
yRegisterHubDiscoveryCallback	25	YVoc	1476-1515
yRegisterLogFunction	26	YVoltage	1519-1558
YRelay	1266-1302	YVSource	1561-1594
ySelectArchitecture	27	YWakeUpMonitor	1598-1633
YSensor	1306-1345	YWakeUpSchedule	1637-1674
YServo	1349-1384	YWatchdog	1678-1723
		YWireless	1726-1756