



## C++ API Reference



# Table of contents

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Using Yocto-Demo with C++ .....</b>	<b>3</b>
2.1. Control of the Led function .....	3
2.2. Control of the module part .....	5
2.3. Error handling .....	7
2.4. Integration variants for the C++ Yoctopuce library .....	8
Blueprint .....	12
<b>3. Reference .....</b>	<b>12</b>
3.1. General functions .....	13
3.2. Accelerometer function interface .....	38
3.3. AnButton function interface .....	84
3.4. CarbonDioxide function interface .....	126
3.5. ColorLed function interface .....	169
3.6. Compass function interface .....	202
3.7. Current function interface .....	246
3.8. DataLogger function interface .....	289
3.9. Formatted data sequence .....	324
3.10. Recorded data sequence .....	334
3.11. Unformatted data sequence .....	347
3.12. Digital IO function interface .....	362
3.13. Display function interface .....	410
3.14. DisplayLayer object interface .....	461
3.15. External power supply control interface .....	493
3.16. Files function interface .....	522
3.17. GenericSensor function interface .....	555
3.18. Gyroscope function interface .....	605
3.19. Yocto-hub port interface .....	660
3.20. Humidity function interface .....	689
3.21. Led function interface .....	732
3.22. LightSensor function interface .....	763
3.23. Magnetometer function interface .....	807
3.24. Measured value .....	853
3.25. Module control interface .....	859

3.26. Network function interface .....	905
3.27. OS control .....	966
3.28. Power function interface .....	993
3.29. Pressure function interface .....	1040
3.30. Pwm function interface .....	1083
3.31. PwmPowerSource function interface .....	1125
3.32. Quaternion interface .....	1152
3.33. Real Time Clock function interface .....	1195
3.34. Reference frame configuration .....	1226
3.35. Relay function interface .....	1266
3.36. Sensor function interface .....	1306
3.37. Servo function interface .....	1349
3.38. Temperature function interface .....	1388
3.39. Tilt function interface .....	1433
3.40. Voc function interface .....	1476
3.41. Voltage function interface .....	1519
3.42. Voltage source function interface .....	1562
3.43. WakeUpMonitor function interface .....	1598
3.44. WakeUpSchedule function interface .....	1637
3.45. Watchdog function interface .....	1678
3.46. Wireless function interface .....	1727
<b>Index .....</b>	<b>1761</b>

# 1. Introduction

This manual is intended to be used as a reference for Yoctopuce C++ 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 Yocto-Demo with C++

C++ is not the simplest language to master. However, if you take care to limit yourself to its essential functionalities, this language can very well be used for short programs quickly coded, and it has the advantage of being easily ported from one operating system to another. Under Windows, all the examples and the project models are tested with Microsoft Visual Studio 2010 Express, freely available on the Microsoft web site<sup>1</sup>. Under Mac OS X, all the examples and project models are tested with XCode 4, available on the App Store. Moreover, under Max OS X and under Linux, you can compile the examples using a command line with GCC using the provided GNUmakefile. In the same manner under Windows, a Makefile allows you to compile examples using a command line, fully knowing the compilation and linking arguments.

Yoctopuce C++ libraries<sup>2</sup> are integrally provided as source files. A section of the low-level library is written in pure C, but you should not need to interact directly with it: everything was done to ensure the simplest possible interaction from C++. The library is naturally also available as binary files, so that you can link it directly if you prefer.

You will soon notice that the C++ API defines many functions which return objects. You do not need to deallocate these objects yourself, the API does it automatically at the end of the application.

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. You will find in the last section of this chapter all the information needed to create a wholly new project linked with the Yoctopuce libraries.

### 2.1. Control of the Led function

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

```
#include "yocto_api.h"
#include "yocto_led.h"

[...]
String errmsg;
YLed *led;

// Get access to your device, connected locally on USB for instance
yRegisterHub("usb", errmsg);
led = yFindLed("YCTOPOC1-123456.led");
```

<sup>1</sup> <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express>

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

```
// 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.

## yocto\_api.h et yocto\_led.h

These two include files provide access to the functions allowing you to manage Yoctopuce modules. `yocto_api.h` must always be used, `yocto_led.h` is necessary to manage modules containing a led, such as Yocto-Demo.

### yRegisterHub

The `yRegisterHub` 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.

### yFindLed

The `yFindLed` 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.

```
YLed *led = yFindLed("YCTOPOC1-123456.led");
YLed *led = yFindLed("YCTOPOC1-123456.MyFunction");
YLed *led = yFindLed("MyModule.led");
YLed *led = yFindLed("MyModule.MyFunction");
YLed *led = yFindLed("MyFunction");
```

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

### isOnline

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

### set\_power

The `set_power()` function of the objet returned by `yFindLed` allows you to turn on and off the led. The argument is `Y_POWER_ON` or `Y_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 your C++ environment and open the corresponding sample project provided in the directory **Examples/Doc-GettingStarted-Yocto-Demo** of the Yoctopuce library. If you prefer to work with your favorite text editor, open the file `main.cpp`, and type `make` to build the example when you are done.

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.

```
#include "yocto_api.h"
#include "yocto_led.h"
#include <iostream>
#include <stdlib.h>

using namespace std;
```

```

static void usage(void)
{
    cout << "usage: demo <serial_number> [ on | off ]" << endl;
    cout << "           demo <logical_name> [ on | off ]" << endl;
    cout << "           demo any [ on | off ]" << endl;
    cout << "                                     (use any discovered device)" <<
endl;
    u64 now = yGetTickCount(); // dirty active wait loop
        while (yGetTickCount()-now<3000);
    exit(1);
}

int main(int argc, const char * argv[])
{
    string errmsg;
    string target;
    YLed *led;
    string on_off;

    if(argc < 3) {
        usage();
    }
    target = (string) argv[1];
    on_off = (string) argv[2];

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

    if(target == "any"){
        led = yFirstLed();
    }else{
        led = yFindLed(target + ".led");
    }
    if (led && led->isOnline()) {
        led->set_power(on_off == "on" ? Y_POWER_ON : Y_POWER_OFF);
    } else {
        cout << "Module not connected (check identification and USB cable)" << endl;
    }

    return 0;
}

```

## 2.2. 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.

```

#include <iostream>
#include <stdlib.h>

#include "yocto_api.h"

using namespace std;

static void usage(const char *exe)
{
    cout << "usage: " << exe << " <serial or logical name> [ON/OFF]" << endl;
    exit(1);
}

int main(int argc, const char * argv[])
{
    string errmsg;

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }
}

```

```

if(argc < 2)
    usage(argv[0]);

YModule *module = yFindModule(argv[1]); // use serial or logical name

if (module->isOnline()) {
    if (argc > 2) {
        if (string(argv[2]) == "ON")
            module->set_beacon(Y_BEACON_ON);
        else
            module->set_beacon(Y_BEACON_OFF);
    }
    cout << "serial:      " << module->get_serialNumber() << endl;
    cout << "logical name: " << module->get_logicalName() << endl;
    cout << "luminosity:   " << module->get_luminosity() << endl;
    cout << "beacon:       ";
    if (module->get_beacon() == Y_BEACON_ON)
        cout << "ON" << endl;
    else
        cout << "OFF" << endl;
    cout << "upTime:      " << module->get_upTime()/1000 << " sec" << endl;
    cout << "USB current: " << module->get_usbCurrent() << " mA" << endl;
    cout << "Logs:" << endl << module->get_lastLogs() << endl;
} else {
    cout << argv[1] << " not connected (check identification and USB cable)"
        << endl;
}
return 0;
}

```

Each property `xxx` of the module can be read thanks to a method of type `get_xxxx()`, and properties which are not read-only can be modified with the help of the `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 `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 `saveToFlash()` method. Inversely, it is possible to force the module to forget its current settings by using the `revertFromFlash()` method. The short example below allows you to modify the logical name of a module.

```

#include <iostream>
#include <stdlib.h>

#include "yocto_api.h"

using namespace std;

static void usage(const char *exe)
{
    cerr << "usage: " << exe << " <serial> <newLogicalName>" << endl;
    exit(1);
}

int main(int argc, const char * argv[])
{
    string      errmsg;

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

    if(argc < 2)
        usage(argv[0]);

    YModule *module = yFindModule(argv[1]); // use serial or logical name

    if (module->isOnline()) {
        if (argc >= 3) {

```

```

        string newname = argv[2];
        if (!yCheckLogicalName(newname)) {
            cerr << "Invalid name (" << newname << ")" << endl;
            usage(argv[0]);
        }
        module->set_logicalName(newname);
        module->saveToFlash();
    }
    cout << "Current name: " << module->get_logicalName() << endl;
} else {
    cout << argv[1] << " not connected (check identification and USB cable)"
    << endl;
}
return 0;
}

```

Warning: the number of write cycles of the nonvolatile memory of the module is limited. When this limit is reached, nothing guarantees 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 `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 `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.

```

#include <iostream>
#include "yocto_api.h"

using namespace std;

int main(int argc, const char * argv[])
{
    string      errmsg;
    // Setup the API to use local USB devices
    if(YAPI::RegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

    cout << "Device list: " << endl;

    YModule *module = YModule::FirstModule();
    while (module != NULL) {
        cout << module->get_serialNumber() << " ";
        cout << module->get_productName() << endl;
        module = module->nextModule();
    }
    return 0;
}

```

## 2.3. 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.

## 2.4. Integration variants for the C++ Yoctopuce library

Depending on your needs and on your preferences, you can integrate the library into your projects in several distinct manners. This section explains how to implement the different options.

### Integration in source format

Integrating all the sources of the library into your projects has several advantages:

- It guarantees the respect of the compilation conventions of your project (32/64 bits, inclusion of debugging symbols, unicode or ASCII characters, etc.);
- It facilitates debugging if you are looking for the cause of a problem linked to the Yoctopuce library;
- It reduces the dependencies on third party components, for example in the case where you would need to recompile this project for another architecture in many years;
- It does not require the installation of a dynamic library specific to Yoctopuce on the final system, everything is in the executable.

To integrate the source code, the easiest way is to simply include the `Sources` directory of your Yoctopuce library into your **IncludePath**, and to add all the files of this directory (including the sub-directory `yapi`) to your project.

For your project to build correctly, you need to link with your project the prerequisite system libraries, that is:

- For Windows: the libraries are added automatically
- For Mac OS X: **IOKit.framework** and **CoreFoundation.framework**
- For Linux: **libm**, **libpthread**, **libusb1.0**, and **libstdc++**

## Integration as a static library

Integration of the Yoctopuce library as a static library is a simpler manner to build a small executable which uses Yoctopuce modules. You can quickly compile the program with a single command. You do not need to install a dynamic library specific to Yoctopuce, everything is in the executable.

To integrate the static Yoctopuce library to your project, you must include the `Sources` directory of the Yoctopuce library into your **IncludePath**, and add the sub-directory `Binaries/...` corresponding to your operating system into your **libPath**.

Then, for you project to build correctly, you need to link with your project the Yoctopuce library and the prerequisite system libraries:

- For Windows: **yocto-static.lib**
- For Mac OS X: **libyocto-static.a**, **IOKit.framework**, and **CoreFoundation.framework**
- For Linux: **libyocto-static.a**, **libm**, **libpthread**, **libusb1.0**, and **libstdc++**.

Note, under Linux, if you wish to compile in command line with GCC, it is generally advisable to link system libraries as dynamic libraries, rather than as static ones. To mix static and dynamic libraries on the same command line, you must pass the following arguments:

```
gcc (...) -Wl,-Bstatic -lyocto-static -Wl,-Bdynamic -lm -lpthread -lusb-1.0 -lstdc++
```

## Integration as a dynamic library

Integration of the Yoctopuce library as a dynamic library allows you to produce an executable smaller than with the two previous methods, and to possibly update this library, if a patch reveals itself necessary, without needing to recompile the source code of the application. On the other hand, it is an integration mode which systematically requires you to copy the dynamic library on the target machine where the application will run (**yocto.dll** for Windows, **libyocto.so.1.0.1** for Mac OS X and Linux).

To integrate the dynamic Yoctopuce library to your project, you must include the `Sources` directory of the Yoctopuce library into your **IncludePath**, and add the sub-directory `Binaries/...` corresponding to your operating system into your **LibPath**.

Then, for you project to build correctly, you need to link with your project the dynamic Yoctopuce library and the prerequisite system libraries:

- For Windows: **yocto.lib**
- For Mac OS X: **libyocto**, **IOKit.framework**, and **CoreFoundation.framework**
- For Linux: **libyocto**, **libm**, **libpthread**, **libusb1.0**, and **libstdc++**.

With GCC, the command line to compile is simply:

```
gcc (...) -lyocto -lm -lpthread -lusb-1.0 -lstdc++
```





### **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>
node.js 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

#### `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()yCheckLogicalName( )**

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

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

Disables the use of exceptions to report runtime errors.

js	function <b>yDisableExceptions( )</b>
node.js	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()yEnableExceptions( )**

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

js	function <b>yEnableExceptions( )</b>
node.js	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()yFreeAPI( )**

Frees dynamically allocated memory blocks used by the Yoctopuce library.

**js** function **yFreeAPI( )****nodejs** function **FreeAPI( )****php** function **yFreeAPI( )****cpp** void **yFreeAPI( )****m** void **yFreeAPI( )****pas** procedure **yFreeAPI( )****vb** procedure **yFreeAPI( )****cs** void **FreeAPI( )****java** void **FreeAPI( )****py** def **FreeAPI( )**

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() yGetAPIVersion()yGetAPIVersion( )

YAPI

Returns the version identifier for the Yoctopuce library in use.

```
js function yGetAPIVersion( )
node.js function GetAPIVersion( )
php function yGetAPIVersion( )
cpp string yGetAPIVersion( )
m NSString* yGetAPIVersion( )
pas function yGetAPIVersion( ): string
vb function yGetAPIVersion( ) As String
cs String GetAPIVersion( )
java String GetAPIVersion( )
py def GetAPIVersion( )
```

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()yGetTickCount( )**

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()yHandleEvents( )**

Maintains the device-to-library communication channel.

```
js function yHandleEvents( errmsg)
node.js function HandleEvents( errmsg)
php function yHandleEvents( &$errmsg)
cpp YRETCODE yHandleEvents( string& errmsg)
m YRETCODE yHandleEvents( NSError** errmsg)
pas function yHandleEvents( var errmsg: string): integer
vb function yHandleEvents( ByRef errmsg As String) As YRETCODE
cs YRETCODE HandleEvents( ref string errmsg)
java int HandleEvents( )
py def HandleEvents( errmsg=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() yInitAPI()yInitAPI( )

YAPI

Initializes the Yoctopuce programming library explicitly.

js	function <b>yInitAPI( mode, errmsg)</b>
nodejs	function <b>InitAPI( mode, errmsg)</b>
php	function <b>yInitAPI( \$mode, &amp;\$errmsg)</b>
cpp	<b>YRETCODE yInitAPI( int mode, string&amp; errmsg)</b>
m	<b>YRETCODE yInitAPI( int mode, NSError** errmsg)</b>
pas	function <b>yInitAPI( mode: integer, var errmsg: string): integer</b>
vb	function <b>yInitAPI( ByVal mode As Integer, ByRef errmsg As String) As Integer</b>
cs	<b>int InitAPI( int mode, ref string errmsg)</b>
java	<b>int InitAPI( int mode)</b>
py	<b>def InitAPI( mode, 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()yPreregisterHub( )

YAPI

Fault-tolerant alternative to RegisterHub().

```
js function yPreregisterHub( url, errmsg)
node.js function PreregisterHub( url, errmsg)
php function yPreregisterHub( $url, &$errmsg)
cpp YRETCODE yPreregisterHub( const string& url, string& errmsg)
m YRETCODE yPreregisterHub( NSString * url, NSError** errmsg)
pas function yPreregisterHub( url: string, var errmsg: string): integer
vb function yPreregisterHub( ByVal url As String,
                           ByRef errmsg As String) As Integer
cs int PreregisterHub( string url, ref string errmsg)
java int PreregisterHub( String url)
py def PreregisterHub( url, errmsg=None)
```

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()****YAPI****yRegisterDeviceArrivalCallback()****yRegisterDeviceArrivalCallback( )**

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

```
js   function yRegisterDeviceArrivalCallback( arrivalCallback)
node.js function RegisterDeviceArrivalCallback( arrivalCallback)
php  function yRegisterDeviceArrivalCallback( $arrivalCallback)
cpp   void yRegisterDeviceArrivalCallback( yDeviceUpdateCallback arrivalCallback)
m    void yRegisterDeviceArrivalCallback( yDeviceUpdateCallback arrivalCallback)
pas   procedure yRegisterDeviceArrivalCallback( arrivalCallback: yDeviceUpdateFunc)
vb    procedure yRegisterDeviceArrivalCallback( ByVal arrivalCallback As yDeviceUpdateFunc)
cs    void RegisterDeviceArrivalCallback( yDeviceUpdateFunc arrivalCallback)
java  void RegisterDeviceArrivalCallback( DeviceArrivalCallback arrivalCallback)
py    def RegisterDeviceArrivalCallback( arrivalCallback)
```

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() yRegisterDeviceRemovalCallback( )

YAPI

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

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

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()****YAPI****yRegisterHub()**

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

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

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()**  
**yRegisterHubDiscoveryCallback( )**

YAPI

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

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

Registers a log callback function.

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

YAPI

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 is not possible to reliably distinguish between a Hard Float (armhf) and a Soft Float (armel) install. For 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. Possible values 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 )
node.js 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()ySleep( )**

Pauses the execution flow for a specified duration.

js	function <b>ySleep( ms_duration, errmsg)</b>
nodejs	function <b>Sleep( ms_duration, errmsg)</b>
php	function <b>ySleep( \$ms_duration, &amp;\$errmsg)</b>
cpp	<b>YRETCODE ySleep( unsigned ms_duration, string&amp; errmsg)</b>
m	<b>YRETCODE ySleep( unsigned ms_duration, NSError ** errmsg)</b>
pas	function <b>ySleep( ms_duration: integer, var errmsg: string): integer</b>
vb	function <b>ySleep( ByVal ms_duration As Integer,</b> <b>ByRef errmsg As String) As Integer</b>
cs	<b>int Sleep( int ms_duration, ref string errmsg)</b>
java	<b>int Sleep( long ms_duration)</b>
py	<b>def Sleep( ms_duration, 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()**

Force a hub discovery, if a callback has 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()yUnregisterHub( )**

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

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

**Parameters :**

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

**YAPI.UpdateDeviceList()**

YAPI

**yUpdateDeviceList()yUpdateDeviceList( )**

Triggers a (re)detection of connected Yoctopuce modules.

```
js function yUpdateDeviceList( errmsg)
node.js function UpdateDeviceList( errmsg)
php function yUpdateDeviceList( &$errmsg)
cpp YRETCODE yUpdateDeviceList( string& errmsg)
m YRETCODE yUpdateDeviceList( NSError** errmsg)
pas function yUpdateDeviceList( var errmsg: string): integer
vb function yUpdateDeviceList( ByRef errmsg As String) As YRETCODE
cs YRETCODE UpdateDeviceList( ref string errmsg)
java int UpdateDeviceList( )
py def UpdateDeviceList( errmsg=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');
cpp #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.

<b>accelerometer→get_highestValue()</b>	Returns the maximal value observed for the acceleration since the device was started.
<b>accelerometer→get_logFrequency()</b>	Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>accelerometer→get_logicalName()</b>	Returns the logical name of the accelerometer.
<b>accelerometer→get_lowestValue()</b>	Returns the minimal value observed for the acceleration since the device was started.
<b>accelerometer→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>accelerometer→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>accelerometer→get_recordedData(startTime, endTime)</b>	Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>accelerometer→get_reportFrequency()</b>	Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>accelerometer→get_resolution()</b>	Returns the resolution of the measured values.
<b>accelerometer→get_unit()</b>	Returns the measuring unit for the acceleration.
<b>accelerometer→get(userData)</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>accelerometer→get_xValue()</b>	Returns the X component of the acceleration, as a floating point number.
<b>accelerometer→get_yValue()</b>	Returns the Y component of the acceleration, as a floating point number.
<b>accelerometer→get_zValue()</b>	Returns the Z component of the acceleration, as a floating point number.
<b>accelerometer→isOnline()</b>	Checks if the accelerometer is currently reachable, without raising any error.
<b>accelerometer→isOnline_async(callback, context)</b>	Checks if the accelerometer is currently reachable, without raising any error (asynchronous version).
<b>accelerometer→load(msValidity)</b>	Preloads the accelerometer cache with a specified validity duration.
<b>accelerometer→loadCalibrationPoints(rawValues, refValues)</b>	Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>accelerometer→load_async(msValidity, callback, context)</b>	Preloads the accelerometer cache with a specified validity duration (asynchronous version).
<b>accelerometer→nextAccelerometer()</b>	Continues the enumeration of accelerometers started using yFirstAccelerometer( ).
<b>accelerometer→registerTimedReportCallback(callback)</b>	Registers the callback function that is invoked on every periodic timed notification.
<b>accelerometer→registerValueCallback(callback)</b>	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()****YAccelerometer****yFindAccelerometer()yFindAccelerometer( )**

Retrieves an accelerometer for a given identifier.

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

Starts the enumeration of accelerometers currently accessible.

```
js function yFirstAccelerometer( )
node.js function FirstAccelerometer( )
php function yFirstAccelerometer( )
cpp YAccelerometer* yFirstAccelerometer( )
m YAccelerometer* yFirstAccelerometer( )
pas function yFirstAccelerometer( ): TYAccelerometer
vb function yFirstAccelerometer( ) As YAccelerometer
cs YAccelerometer FirstAccelerometer( )
java YAccelerometer FirstAccelerometer( )
py def FirstAccelerometer( )
```

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 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  YAccelerometer 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.

**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	- <b>(NSString*) 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	<b>def 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)=RELAYL01-123456.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 accelerometer (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.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>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
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	<b>def get_advertisedValue( )</b>
cmd	YAccelerometer <b>target 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()**  
**accelerometer→currentRawValue()accelerometer**  
**→get\_currentRawValue()**

**YAccelerometer**

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

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

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

**YAccelerometer**

Returns the current value of the acceleration.

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

**YAccelerometer**

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

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

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

**Returns :**

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

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

**YAccelerometer**

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

js	function <b>get_errorType( )</b>
node.js	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	<b>YRETCODE get_errorType( )</b>
pas	function <b>get_errorType( ): YRETCODE</b>
vb	function <b>get_errorType( ) As YRETCODE</b>
cs	<b>YRETCODE get_errorType( )</b>
java	<b>int get_errorType( )</b>
py	<b>def 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 get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the accelerometer if they are defined, otherwise the serial number of the module and the hardware identifier of the accelerometer (for exemple: 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 get_functionDescriptor( )  
node.js 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.

**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>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the accelerometer (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

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

YAccelerometer

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

js    function **get\_highestValue( )**  
nodejs    function **get\_highestValue( )**  
php    function **get\_highestValue( )**  
cpp    double **get\_highestValue( )**  
m    -(double) highestValue  
pas    function **get\_highestValue( )**: double  
vb    function **get\_highestValue( )** As Double  
cs    double **get\_highestValue( )**  
java    double **get\_highestValue( )**  
py    def **get\_highestValue( )**  
cmd    YAccelerometer **target get\_highestValue**

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

**Returns :**

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

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

**YAccelerometer**

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

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

**YAccelerometer**

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	<b>YModule * get_module( )</b>
m	<b>-(YModule*) module</b>
pas	function <b>get_module( )</b> : TYModule
vb	function <b>get_module( )</b> As <b>YModule</b>
cs	<b>YModule get_module( )</b>
java	<b>YModule get_module( )</b>
py	<b>def 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()**  
**accelerometer→module\_async()****YAccelerometer**

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

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

**YAccelerometer**

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

js	function <b>get_recordedData( startTime, endTime)</b>
nodejs	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	YDataSet <b>get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
java	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
py	def <b>get_recordedData( startTime, endTime)</b>
cmd	<b>YAccelerometer target get_recordedData startTime 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 measure, 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 measure, 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 get_reportFrequency( )
nodejs function get_reportFrequency( )
php  function get_reportFrequency( )
cpp   string get_reportFrequency( )
m    -(NSString*) reportFrequency
pas   function get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs   string get_reportFrequency( )
java  String get_reportFrequency( )
py    def get_reportFrequency( )
cmd   YAccelerometer target get_reportFrequency
```

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

YAccelerometer

Returns the resolution of the measured values.

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

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*) <b>unit</b>
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	<b>YAccelerometer target 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 get(userData) 
nodejs function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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

**Returns :**

the object stored previously by the caller.

**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>
node.js	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 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()  
accelerometer→yValue()accelerometer→  
get\_yValue()

YAccelerometer

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

js    function get\_yValue( )  
nodejs function get\_yValue( )  
php    function get\_yValue( )  
cpp    double get\_yValue( )  
m      -(double) yValue  
pas    function get\_yValue( ): double  
vb     function get\_yValue( ) As Double  
cs     double get\_yValue( )  
java    double get\_yValue( )  
py     def get\_yValue( )  
cmd    YAccelerometer target get\_yValue

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

**YAccelerometer**

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

<b>js</b>	function <b>get_zValue( )</b>
<b>node.js</b>	function <b>get_zValue( )</b>
<b>php</b>	function <b>get_zValue( )</b>
<b>cpp</b>	double <b>get_zValue( )</b>
<b>m</b>	-(double) zValue
<b>pas</b>	function <b>get_zValue( ): double</b>
<b>vb</b>	function <b>get_zValue( ) As Double</b>
<b>cs</b>	double <b>get_zValue( )</b>
<b>java</b>	double <b>get_zValue( )</b>
<b>py</b>	<b>def get_zValue( )</b>
<b>cmd</b>	<b>YAccelerometer target 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 isOnline( )
node.js function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

If there is a cached value for the 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 <b>isOnline_async( callback, context)</b>
node.js	function <b>isOnline_async( callback, context)</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 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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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()

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()**  
→**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()****YAccelerometer****accelerometer→****registerTimedReportCallback( )**

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

<b>js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>nodejs</b>	function registerTimedReportCallback( <b>callback</b> )
<b>php</b>	function registerTimedReportCallback( <b>\$callback</b> )
<b>cpp</b>	int registerTimedReportCallback( YAccelerometerTimedReportCallback <b>callback</b> )
<b>m</b>	- <b>(int) registerTimedReportCallback : (YAccelerometerTimedReportCallback) callback</b>
<b>pas</b>	function registerTimedReportCallback( <b>callback</b> : TYAccelerometerTimedReportCallback): LongInt
<b>vb</b>	function registerTimedReportCallback( ) As Integer
<b>cs</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>java</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>py</b>	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YAccelerometerValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YAccelerometerValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYAccelerometerValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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 set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YAccelerometer target set_highestValue newval
```

**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.

---

<b>accelerometer→set_logFrequency()</b> <b>accelerometer→setLogFrequency()accelerometer</b> <b>→set_logFrequency( )</b>	<b>YAccelerometer</b>
---	-----------------------

---

Changes the datalogger recording frequency for this function.

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

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

Changes the logical name of the accelerometer.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the accelerometer.

**Returns :**

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

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

Changes the recorded minimal value observed.

```
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   YAccelerometer target set_lowestValue newval
```

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

YAccelerometer

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   YAccelerometer 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.

---

**accelerometer→set\_resolution()** YAccelerometer

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

---

Changes the resolution of the measured physical values.

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

**accelerometer→set(userData)**  
**accelerometer→setUserData()accelerometer→**  
**set(userData)**

YAccelerometer

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

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

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

**Parameters :**

**data** any kind of object to be stored

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

```

#### 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()****YAnButton****yFindAnButton()yFindAnButton( )**

Retrieves an analog input for a given identifier.

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

Starts the enumeration of analog inputs currently accessible.

js	function <b>yFirstAnButton( )</b>
node.js	function <b>FirstAnButton( )</b>
php	function <b>yFirstAnButton( )</b>
cpp	<b>YAnButton*</b> <b>yFirstAnButton( )</b>
m	<b>YAnButton*</b> <b>yFirstAnButton( )</b>
pas	function <b>yFirstAnButton( )</b> : TYAnButton
vb	function <b>yFirstAnButton( )</b> As <b>YAnButton</b>
cs	<b>YAnButton FirstAnButton( )</b>
java	<b>YAnButton 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()****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** 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1`)

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

**YAnButton**

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

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

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

**YAnButton**

Tells if a calibration process is currently ongoing.

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

**YAnButton**

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

**js** function **get\_calibratedValue( )**  
**nodejs** function **get\_calibratedValue( )**  
**php** function **get\_calibratedValue( )**  
**cpp** int **get\_calibratedValue( )**  
**m** -(int) calibratedValue  
**pas** function **get\_calibratedValue( )**: LongInt  
**vb** function **get\_calibratedValue( )** As Integer  
**cs** int **get\_calibratedValue( )**  
**java** int **get\_calibratedValue( )**  
**py** def **get\_calibratedValue( )**  
**cmd** YAnButton target **get\_calibratedValue**

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

**YAnButton**

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

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

**YAnButton**

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

**js** function **get\_calibrationMin( )**  
**nodejs** function **get\_calibrationMin( )**  
**php** function **get\_calibrationMin( )**  
**cpp** int **get\_calibrationMin( )**  
**m** -(int) calibrationMin  
**pas** function **get\_calibrationMin( )**: LongInt  
**vb** function **get\_calibrationMin( )** As Integer  
**cs** int **get\_calibrationMin( )**  
**java** int **get\_calibrationMin( )**  
**py** def **get\_calibrationMin( )**  
**cmd** YAnButton target **get\_calibrationMin**

**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→getErrorMessage()**  
**anbutton→errorMessage()** anbutton→  
**getErrorMessage()**

**YAnButton**

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

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

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

**YAnButton**

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

**js** function **get\_errorType( )**  
**nodejs** function **get\_errorType( )**  
**php** function **get\_errorType( )**  
**cpp** YRETCODE **get\_errorType( )**  
**pas** function **get\_errorType( )**: YRETCODE  
**vb** function **get\_errorType( )** As YRETCODE  
**cs** YRETCODE **get\_errorType( )**  
**java** int **get\_errorType( )**  
**py** def **get\_errorType( )**

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the analog input object

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

**YAnButton**

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

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

The returned string uses the logical names of the module and of the 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()**  
**anbutton→functionDescriptor()** anbutton→  
**get\_functionDescriptor( )**

**YAnButton**

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

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	String <b>get_functionDescriptor( )</b>
<b>py</b>	<b>def 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()**  
**anbutton→functionId()** anbutton→  
**get\_functionId( )**

**YAnButton**

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

js	function <b>get_functionId( )</b>
node.js	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( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def 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()**  
**anbutton→hardwareId()** anbutton→  
**get\_hardwareId( )**

**YAnButton**

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

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the analog input. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the analog input (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

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

**YAnButton**

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

js	function <b>get_isPressed( )</b>
node.js	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 target <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()**  
**anbutton→lastTimePressed()****anbutton→get\_lastTimePressed( )**

**YAnButton**

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

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

**YAnButton**

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

**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\_LASTTIMERELEASED\_INVALID**.

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

**YAnButton**

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 target <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>
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 online.

**Returns :**

an instance of YModule

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

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

`anbutton->get_pulseCounter()`  
`anbutton->pulseCounter()``anbutton->`  
`get_pulseCounter()`

`YAnButton`

Returns the pulse counter value

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

**Returns :**

an integer corresponding to the pulse counter value

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

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

**YAnButton**

Returns the timer of the pulses counter (ms)

**js** function **get\_pulseTimer( )**  
**nodejs** function **get\_pulseTimer( )**  
**php** function **get\_pulseTimer( )**  
**cpp** s64 **get\_pulseTimer( )**  
**m** -(s64) pulseTimer  
**pas** function **get\_pulseTimer( )**: int64  
**vb** function **get\_pulseTimer( )** As Long  
**cs** long **get\_pulseTimer( )**  
**java** long **get\_pulseTimer( )**  
**py** def **get\_pulseTimer( )**

**Returns :**

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

On failure, throws an exception or returns **Y\_PULSE\_TIMER\_INVALID**.

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

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 target <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()  
anbutton→sensitivity()anbutton→  
get\_sensitivity()****YAnButton**

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

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()****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	- <b>(BOOL) 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	<b>def 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)
node.js	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()****YAnButton**

Preloads the analog input cache with a specified validity duration.

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

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

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

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

**Parameters :**

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

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

Returns the pulse counter value as well as his timer

js	function <b>resetCounter( )</b>
node.js	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.

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

**YAnButton**

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

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

**YAnButton**

Changes the logical name of the analog input.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the analog input.

**Returns :**

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

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

**YAnButton**

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

js	function <b>set_sensitivity( newval)</b>
node.js	function <b>set_sensitivity( newval)</b>
php	function <b>set_sensitivity( \$newval)</b>
cpp	int <b>set_sensitivity( int newval)</b>
m	-(int) setSensitivity : (int) <b>newval</b>
pas	function <b>set_sensitivity( newval: LongInt): integer</b>
vb	function <b>set_sensitivity( ByVal newval As Integer) As Integer</b>
cs	int <b>set_sensitivity( int newval)</b>
java	int <b>set_sensitivity( int newval)</b>
py	def <b>set_sensitivity( 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)**  
**anbutton→setUserData()**  
**anbutton→set(userData)**

**YAnButton**

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

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b>
m	-(void) <b>set(userData)</b>
pas	procedure <b>set(userData)</b>
vb	procedure <b>set(userData)</b>
cs	void <b>set(userData)</b>
java	void <b>set(userData)</b>
py	def <b>set(userData)</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');
cpp #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()yFindCarbonDioxide( )****YCarbonDioxide**

Retrieves a CO2 sensor for a given identifier.

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

Starts the enumeration of CO2 sensors currently accessible.

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

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()****YCarbonDioxide****carbondioxide→calibrateFromPoints()**

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  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.

**carbon dioxide → describe() carbon dioxide →  
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 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)=RELAYL01-123456.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 CO2 sensor (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.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>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
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 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.

**carbon dioxide → get\_currentRawValue()**  
**carbon dioxide → currentRawValue() carbon dioxide**  
**→ get\_currentRawValue()**

**YCarbonDioxide**

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

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( )**: double  
**vb** function **get\_currentRawValue( )** As Double  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YCarbonDioxide **target get\_currentRawValue**

**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) <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	YCarbonDioxide <b>target 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()carbon dioxide→**  
**get\_errorMessage( )**

**YCarbonDioxide**

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

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

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

**Returns :**

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

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

---

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

js	function <b>get_errorType( )</b>
node.js	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	<b>YRETCODE get_errorType( )</b>
pas	function <b>get_errorType( ): YRETCODE</b>
vb	function <b>get_errorType( ) As YRETCODE</b>
cs	<b>YRETCODE 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

**carbon dioxide → get\_friendlyName()** **YCarbonDioxide**  
**carbon dioxide → friendlyName() carbon dioxide →**  
**get\_friendlyName( )**

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

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

---

<b>carbondioxide→get_functionDescriptor()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→functionDescriptor()carbondioxide</b> <b>→get_functionDescriptor( )</b>	

---

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.

**carbon dioxide → get\_functionId()**  
**carbon dioxide → functionId()** carbon dioxide →  
**get\_functionId( )**

**YCarbonDioxide**

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

js	function <b>get_functionId( )</b>
node.js	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()****YCarbonDioxide****carbondioxide→hardwareId() carbondioxide→  
get.hardwareId( )**

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

js	function <b>get.hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the CO2 sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

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

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 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()****YCarbonDioxide****carbondioxide→logFrequency() carbondioxide→  
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	<b>YCarbonDioxide target 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**.

**carbon dioxide → get\_logicalName()** **YCarbonDioxide**  
**carbon dioxide → logicalName()** **carbon dioxide →**  
**get\_logicalName( )**

Returns the logical name of the CO2 sensor.

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

**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()**  
**carbondioxide→lowestValue()carbon dioxide→**  
**get\_lowestValue( )**

**YCarbonDioxide**

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

js	function <b>get_lowestValue( )</b>
node.js	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 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.

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

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**carbon dioxide → get\_recordedData()**  
**carbon dioxide → recordedData() carbon dioxide →**  
**get\_recordedData( )**

**YCarbonDioxide**

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

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

---

<b>carbondioxide→get_reportFrequency()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→reportFrequency()carbon dioxide→</b> <b>get_reportFrequency( )</b>	

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

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

**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	<b>YCarbonDioxide target 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()carbon dioxide→get\_unit()**

Returns the measuring unit for the CO2 concentration.

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

**Returns :**

a string corresponding to the measuring unit for the CO2 concentration

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

**carbon dioxide → get(userData)****YCarbonDioxide****carbon dioxide → userData() carbon dioxide →****get(userData())**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

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

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

**Returns :**

the object stored previously by the caller.

**carbondioxide→isOnline()carbon dioxide→  
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()****YCarbonDioxide**

Preloads the CO2 sensor cache with a specified validity duration.

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

**carbondioxide→loadCalibrationPoints()****YCarbonDioxide****carbondioxide→loadCalibrationPoints( )**

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()`.

<code>js</code>	<b>function nextCarbonDioxide( )</b>
<code>node.js</code>	<b>function nextCarbonDioxide( )</b>
<code>php</code>	<b>function nextCarbonDioxide( )</b>
<code>cpp</code>	<b>YCarbonDioxide * nextCarbonDioxide( )</b>
<code>m</code>	<b>-(YCarbonDioxide*) nextCarbonDioxide</b>
<code>pas</code>	<b>function nextCarbonDioxide( ): TYCarbonDioxide</b>
<code>vb</code>	<b>function nextCarbonDioxide( ) As YCarbonDioxide</b>
<code>cs</code>	<b>YCarbonDioxide nextCarbonDioxide( )</b>
<code>java</code>	<b>YCarbonDioxide nextCarbonDioxide( )</b>
<code>py</code>	<b>def 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()****YCarbonDioxide****carbondioxide→****registerTimedReportCallback( )**

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

<b>js</b>	<code>function registerTimedReportCallback( callback)</code>
<b>node.js</b>	<code>function registerTimedReportCallback( callback)</code>
<b>php</b>	<code>function registerTimedReportCallback( \$callback)</code>
<b>cpp</b>	<code>int registerTimedReportCallback( YCarbonDioxideTimedReportCallback callback)</code>
<b>m</b>	<code>-(int) registerTimedReportCallback : (YCarbonDioxideTimedReportCallback) callback</code>
<b>pas</b>	<code>function registerTimedReportCallback( callback: TYCarbonDioxideTimedReportCallback): LongInt</code>
<b>vb</b>	<code>function registerTimedReportCallback( ) As Integer</code>
<b>cs</b>	<code>int registerTimedReportCallback( TimedReportCallback callback)</code>
<b>java</b>	<code>int registerTimedReportCallback( TimedReportCallback callback)</code>
<b>py</b>	<code>def registerTimedReportCallback( callback)</code>

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.

**carbon dioxide → registerValueCallback()****YCarbonDioxide****carbon dioxide → registerValueCallback( )**

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

```
js   function registerValueCallback( callback )
node.js function registerValueCallback( callback )
php  function registerValueCallback( $callback )
cpp   int registerValueCallback( YCarbonDioxideValueCallback callback )
m    -(int) registerValueCallback : (YCarbonDioxideValueCallback) callback
pas   function registerValueCallback( callback: TYCarbonDioxideValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs    int registerValueCallback( ValueCallback callback )
java  int registerValueCallback( UpdateCallback callback )
py    def registerValueCallback( callback )
```

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

**Parameters :**

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

**carbondioxide→set\_highestValue()****YCarbonDioxide****carbondioxide→setHighestValue()|carbondioxide→  
set\_highestValue()**

Changes the recorded maximal value observed.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YCarbonDioxide target set_highestValue newval
```

**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.

**carbon dioxide → set\_logFrequency()** **YCarbonDioxide**  
**carbon dioxide → setLogFrequency() carbon dioxide**  
**→ set\_logFrequency( )**

Changes the datalogger recording frequency for this function.

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

<b>carbondioxide→set_logicalName()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→setLogicalName()carbon dioxide→</b> <b>set_logicalName( )</b>	

Changes the logical name of the CO2 sensor.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YCarbonDioxide 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 CO2 sensor.

#### Returns :

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

**carbon dioxide → set\_lowestValue()** **YCarbonDioxide**  
**carbon dioxide → setLowestValue()** **carbon dioxide →**  
**set\_lowestValue()**

Changes the recorded minimal value observed.

```
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 YCarbonDioxide target set_lowestValue newval
```

**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.

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

**carbon dioxide → set\_resolution()** **YCarbonDioxide**  
**carbon dioxide → setResolution()** **carbon dioxide →**  
**set\_resolution()**

Changes the resolution of the measured physical values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YCarbonDioxide target set_resolution newval
```

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)****YCarbonDioxide****carbondioxide→setUserData() carbondioxide→****set(userData( )**

---

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

js	function <b>set(userData( data)</b>
node.js	function <b>set(userData( data)</b>
php	function <b>set(userData( \$data)</b>
cpp	void <b>set(userData( void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData( data: Tobject)</b>
vb	procedure <b>set(userData( ByVal data As Object)</b>
cs	void <b>set(userData( object data)</b>
java	void <b>set(userData( Object data)</b>
py	def <b>set(userData( 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	<script type='text/javascript' src='yocto_colorled.js'></script>
node.js	var yoctolib = require('yoctolib');
php	var YColorLed = yoctolib.YColorLed;
cpp	require_once('yocto_colorled.php');
m	#include "yocto_colorled.h"
pas	#import "yocto_colorled.h"
vb	uses yocto_colorled;
cs	yocto_colorled.vb
java	yocto_colorled.cs
py	import com.yoctopuce.YoctoAPI.YColorLed;
	from yocto_colorled import *

### 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.

### 3. Reference

<b>colorled→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>colorled→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>colorled→get_rgbColor()</b>	Returns the current RGB color of the led.
<b>colorled→get_rgbColorAtPowerOn()</b>	Returns the configured color to be displayed when the module is turned on.
<b>colorled→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>colorled→hslMove(hsl_target, ms_duration)</b>	Performs a smooth transition in the HSL color space between the current color and a target color.
<b>colorled→isOnline()</b>	Checks if the RGB led is currently reachable, without raising any error.
<b>colorled→isOnline_async(callback, context)</b>	Checks if the RGB led is currently reachable, without raising any error (asynchronous version).
<b>colorled→load(msValidity)</b>	Preloads the RGB led cache with a specified validity duration.
<b>colorled→load_async(msValidity, callback, context)</b>	Preloads the RGB led cache with a specified validity duration (asynchronous version).
<b>colorled→nextColorLed()</b>	Continues the enumeration of RGB leds started using yFirstColorLed( ).
<b>colorled→registerValueCallback(callback)</b>	Registers the callback function that is invoked on every change of advertised value.
<b>colorled→rgbMove(rgb_target, ms_duration)</b>	Performs a smooth transition in the RGB color space between the current color and a target color.
<b>colorled→set_hslColor(newval)</b>	Changes the current color of the led, using a color HSL.
<b>colorled→set_logicalName(newval)</b>	Changes the logical name of the RGB led.
<b>colorled→set_rgbColor(newval)</b>	Changes the current color of the led, using a RGB color.
<b>colorled→set_rgbColorAtPowerOn(newval)</b>	Changes the color that the led will display by default when the module is turned on.
<b>colorled→set_userData(data)</b>	Stores a user context provided as argument in the userData attribute of the function.
<b>colorled→wait_async(callback, context)</b>	Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

**YColorLed.FindColorLed()****yFindColorLed()yFindColorLed( )****YColorLed**

Retrieves an RGB led for a given identifier.

js	function <b>yFindColorLed( func)</b>
node.js	function <b>FindColorLed( func)</b>
php	function <b>yFindColorLed( \$func)</b>
cpp	YColorLed* <b>yFindColorLed( const string&amp; func)</b>
m	YColorLed* <b>yFindColorLed( NSString* func)</b>
pas	function <b>yFindColorLed( func: string): TYColorLed</b>
vb	function <b>yFindColorLed( ByVal func As String) As YColorLed</b>
cs	YColorLed <b>FindColorLed( string func)</b>
java	YColorLed <b>FindColorLed( String func)</b>
py	def <b>FindColorLed( 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()yFirstColorLed( )**

Starts the enumeration of RGB leds currently accessible.

js	function <b>yFirstColorLed( )</b>
node.js	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()****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)=RELAYL01-123456.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 RGB led (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**colorled→get\_advertisedValue()**  
**colorled→advertisedValue()colorled→**  
**get\_advertisedValue( )****YColorLed**

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

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

**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→getErrorMessage()**  
**colorled→errorMessage()colorled→**  
**getErrorMessage( )**

**YColorLed**

Returns the error message of the latest error with the RGB led.

<b>js</b>	function <b>getErrorMessage( )</b>
<b>nodejs</b>	function <b>getErrorMessage( )</b>
<b>php</b>	function <b>getErrorMessage( )</b>
<b>cpp</b>	string <b>getErrorMessage( )</b>
<b>m</b>	-(NSString*) errorMessage
<b>pas</b>	function <b>getErrorMessage( )</b> : string
<b>vb</b>	function <b>getErrorMessage( )</b> As String
<b>cs</b>	string <b>getErrorMessage( )</b>
<b>java</b>	String <b>getErrorMessage( )</b>
<b>py</b>	def <b>getErrorMessage( )</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>
node.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 the RGB led object

**colorled→get\_friendlyName()**  
**colorled→friendlyName()colorled→**  
**get\_friendlyName( )**

**YColorLed**

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

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

The returned string uses the logical names of the module and of the 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.

**colorled→get\_functionDescriptor()**  
**colorled→functionDescriptor()colorled→**  
**get\_functionDescriptor( )**

**YColorLed**

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

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	String <b>get_functionDescriptor( )</b>
<b>py</b>	<b>def 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()**  
**colorled→functionId()colorled→**  
**get\_functionId( )**

**YColorLed**

Returns the hardware identifier of the RGB led, without reference to the module.

js	function <b>get_functionId( )</b>
node.js	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( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def 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`.

**colorled→get\_hardwareId()**  
**colorled→hardwareId()colorled→**  
**get\_hardwareId( )**

**YColorLed**

Returns the unique hardware identifier of the RGB led in the form SERIAL.FUNCTIONID.

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the RGB led. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the RGB led (ex: RELAYL01-123456.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 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.

**colorled→get\_logicalName()  
colorled→logicalName()colorled→  
get\_logicalName( )****YColorLed**

Returns the logical name of the RGB led.

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

**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> : 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 online.

**Returns :**

an instance of YModule

**colorled→get\_module\_async()**  
**colorled→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 )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**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) <b>rgbColor</b>
pas	function <b>get_rgbColor( ): LongInt</b>
vb	function <b>get_rgbColor( ) As Integer</b>
cs	int <b>get_rgbColor( )</b>
java	int <b>get_rgbColor( )</b>
py	def <b>get_rgbColor( )</b>
cmd	<b>YColorLed target 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**.

**colorled→get\_rgbColorAtPowerOn()**  
**colorled→rgbColorAtPowerOn()** colorled→  
**get\_rgbColorAtPowerOn( )**

**YColorLed**

Returns the configured color to be displayed when the module is turned on.

**js** function get\_rgbColorAtPowerOn( )  
**nodejs** function get\_rgbColorAtPowerOn( )  
**php** function get\_rgbColorAtPowerOn( )  
**cpp** int get\_rgbColorAtPowerOn( )  
**m** -(int) rgbColorAtPowerOn  
**pas** function get\_rgbColorAtPowerOn( ): LongInt  
**vb** function get\_rgbColorAtPowerOn( ) As Integer  
**cs** int get\_rgbColorAtPowerOn( )  
**java** int get\_rgbColorAtPowerOn( )  
**py** def get\_rgbColorAtPowerOn( )  
**cmd** YColorLed target get\_rgbColorAtPowerOn

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

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.

**colorled→hslMove()colorled→hslMove( )****YColorLed**

Performs a smooth transition in the HSL color space between the current color and a target color.

```
js function hslMove( hsl_target, ms_duration)
nodejs function hslMove( hsl_target, ms_duration)
php function hslMove( $hsl_target, $ms_duration)
cpp int hslMove( int hsl_target, int ms_duration)
m -(int) hslMove : (int) hsl_target : (int) ms_duration
pas function hslMove( hsl_target: LongInt, ms_duration: LongInt): integer
vb function hslMove( ByVal hsl_target As Integer,
                     ByVal ms_duration As Integer) As Integer
cs int hslMove( int hsl_target, int ms_duration)
java int hslMove( int hsl_target, int ms_duration)
py def hslMove( hsl_target, ms_duration)
cmd YColorLed target hslMove hsl_target ms_duration
```

**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()****YColorLed**

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

js	function <b>isOnline( )</b>
node.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 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.

**colorled→load()colorled→load( )****YColorLed**

Preloads the RGB led cache with a specified validity duration.

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

## 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()`.

<code>js</code>	<code>function nextColorLed()</code>
<code>nodejs</code>	<code>function nextColorLed()</code>
<code>php</code>	<code>function nextColorLed()</code>
<code>cpp</code>	<code>YColorLed * nextColorLed()</code>
<code>m</code>	<code>-(YColorLed*) nextColorLed</code>
<code>pas</code>	<code>function nextColorLed(): TYColorLed</code>
<code>vb</code>	<code>function nextColorLed() As YColorLed</code>
<code>cs</code>	<code>YColorLed nextColorLed()</code>
<code>java</code>	<code>YColorLed nextColorLed()</code>
<code>py</code>	<code>def nextColorLed()</code>

**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.

**colorled→registerValueCallback()colorled→registerValueCallback( )****YColorLed**

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

```
js   function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php  function registerValueCallback( $callback)
cpp   int registerValueCallback( YColorLedValueCallback callback)
m    -(int) registerValueCallback : (YColorLedValueCallback) callback
pas   function registerValueCallback( callback: TYColorLedValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs    int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
py    def registerValueCallback( callback)
```

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

**Parameters :**

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

**colorled→rgbMove()****YColorLed**

Performs a smooth transition in the RGB color space between the current color and a target color.

<b>js</b>	function <b>rgbMove(</b> <b>rgb_target, ms_duration)</b>
<b>nodejs</b>	function <b>rgbMove(</b> <b>rgb_target, ms_duration)</b>
<b>php</b>	function <b>rgbMove(</b> \$rgb_target, \$ms_duration)
<b>cpp</b>	int <b>rgbMove(</b> int <b>rgb_target, int ms_duration)</b>
<b>m</b>	-( <b>int</b> ) <b>rgbMove : (int) rgb_target : (int) ms_duration</b>
<b>pas</b>	function <b>rgbMove(</b> <b>rgb_target: LongInt, ms_duration: LongInt): integer</b>
<b>vb</b>	function <b>rgbMove(</b> <b>ByVal rgb_target As Integer,</b> <b>ByVal ms_duration As Integer) As Integer</b>
<b>cs</b>	int <b>rgbMove(</b> int <b>rgb_target, int ms_duration)</b>
<b>java</b>	int <b>rgbMove(</b> int <b>rgb_target, int ms_duration)</b>
<b>py</b>	def <b>rgbMove(</b> <b>rgb_target, ms_duration)</b>
<b>cmd</b>	<b>YColorLed target</b> <b>rgbMove</b> <b>rgb_target ms_duration</b>

**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.

**colorled→set\_hslColor()****YColorLed****colorled→setHslColor()colorled→set\_hslColor( )**

Changes the current color of the led, using a color HSL.

js	function <b>set_hslColor( newval)</b>
node.js	function <b>set_hslColor( newval)</b>
php	function <b>set_hslColor( \$newval)</b>
cpp	int <b>set_hslColor( int newval)</b>
m	-{int) setHslColor : (int) <b>newval</b>
pas	function <b>set_hslColor( newval: LongInt): integer</b>
vb	function <b>set_hslColor( ByVal newval As Integer) As Integer</b>
cs	int <b>set_hslColor( int newval)</b>
java	int <b>set_hslColor( int newval)</b>
py	def <b>set_hslColor( newval)</b>
cmd	<b>YColorLed target set_hslColor newval</b>

Encoding is done as follows: 0xHHSSLL.

**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()**  
**colorled→setLogicalName()** colorled→  
**set\_logicalName( )**

**YColorLed**

Changes the logical name of the RGB led.

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

**Returns :**

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

**colorled→set\_rgbColor()  
colorled→setRgbColor()colorled→  
set\_rgbColor()****YColorLed**

Changes the current color of the led, using a RGB color.

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

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( newval)</b>
node.js	function <b>set_rgbColorAtPowerOn( newval)</b>
php	function <b>set_rgbColorAtPowerOn( \$newval)</b>
cpp	int <b>set_rgbColorAtPowerOn( int newval)</b>
m	- <b>(int) setRgbColorAtPowerOn : (int) newval</b>
pas	function <b>set_rgbColorAtPowerOn( newval: LongInt): integer</b>
vb	function <b>set_rgbColorAtPowerOn( ByVal newval As Integer) As Integer</b>
cs	int <b>set_rgbColorAtPowerOn( int newval)</b>
java	int <b>set_rgbColorAtPowerOn( int newval)</b>
py	def <b>set_rgbColorAtPowerOn( newval)</b>
cmd	YColorLed <b>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.

**colorled→set(userData)**  
**colorled→setUserData()** colorled→  
**set(userData)****YColorLed**

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

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</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');
cpp #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)**

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()yFindCompass( )

**YCompass**

Retrieves a compass for a given identifier.

<code>js</code>	<code>function yFindCompass( func)</code>
<code>nodejs</code>	<code>function FindCompass( func)</code>
<code>php</code>	<code>function yFindCompass( \$func)</code>
<code>cpp</code>	<code>YCompass* yFindCompass( const string&amp; func)</code>
<code>m</code>	<code>YCompass* yFindCompass( NSString* func)</code>
<code>pas</code>	<code>function yFindCompass( func: string): TYCompass</code>
<code>vb</code>	<code>function yFindCompass( ByVal func As String) As YCompass</code>
<code>cs</code>	<code>YCompass FindCompass( string func)</code>
<code>java</code>	<code>YCompass FindCompass( String func)</code>
<code>py</code>	<code>def FindCompass( 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 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()**

Starts the enumeration of compasses currently accessible.

js	function <b>yFirstCompass()</b>
node.js	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 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  YCompass 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.

**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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**compass→get\_advertisedValue()**  
**compass→advertisedValue()compass→**  
**get\_advertisedValue( )**

**YCompass**

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

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) advertisedValue
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YCompass target 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 **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( ): double**  
**vb** function **get\_currentRawValue( ) As Double**  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YCompass **target get\_currentRawValue**

**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()**  
**compass→currentValue()** compass→  
**get\_currentValue()**

**YCompass**

Returns the current value of the relative bearing.

<b>js</b>	function <b>get_currentValue( )</b>
<b>nodejs</b>	function <b>get_currentValue( )</b>
<b>php</b>	function <b>get_currentValue( )</b>
<b>cpp</b>	double <b>get_currentValue( )</b>
<b>m</b>	-(double) <b>currentValue</b>
<b>pas</b>	function <b>get_currentValue( )</b> : double
<b>vb</b>	function <b>get_currentValue( )</b> As Double
<b>cs</b>	double <b>get_currentValue( )</b>
<b>java</b>	double <b>get_currentValue( )</b>
<b>py</b>	def <b>get_currentValue( )</b>
<b>cmd</b>	YCompass <b>target 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()**  
**compass→errorMessage()**compass→  
**get\_errorMessage( )**

**YCompass**

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

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

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

**Returns :**

a string corresponding to the latest error message that 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( )**

**YCompass**

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

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

The returned string uses the logical names of the module and of the 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>
node.js	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.

**compass→get\_functionId()**  
**compass→functionId()compass→**  
**get\_functionId( )**

**YCompass**

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()**  
**compass→hardwareId()compass→**  
**get\_hardwareId( )**

**YCompass**

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

js	function <b>get_hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the compass (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**compass→get\_highestValue()**  
**compass→highestValue()** compass→  
**get\_highestValue( )**

**YCompass**

Returns the maximal value observed for the relative bearing since the device was started.

js    function **get\_highestValue( )**  
nodejs    function **get\_highestValue( )**  
php    function **get\_highestValue( )**  
cpp    double **get\_highestValue( )**  
m    -(double) highestValue  
pas    function **get\_highestValue( )**: double  
vb    function **get\_highestValue( )** As Double  
cs    double **get\_highestValue( )**  
java    double **get\_highestValue( )**  
py    def **get\_highestValue( )**  
cmd    YCompass **target get\_highestValue**

**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()**  
**compass→logFrequency()compass→**  
**get\_logFrequency( )**

**YCompass**

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

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

**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()**  
**compass→logicalName()** compass→  
**get\_logicalName( )**

**YCompass**

Returns the logical name of the compass.

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

**Returns :**

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

**compass→get\_lowestValue()**  
**compass→lowestValue()compass→**  
**get\_lowestValue( )**

**YCompass**

Returns the minimal value observed for the relative bearing since the device was started.

<b>js</b>	function <b>get_lowestValue( )</b>
<b>nodejs</b>	function <b>get_lowestValue( )</b>
<b>php</b>	function <b>get_lowestValue( )</b>
<b>cpp</b>	double <b>get_lowestValue( )</b>
<b>m</b>	-(double) lowestValue
<b>pas</b>	function <b>get_lowestValue( ): double</b>
<b>vb</b>	function <b>get_lowestValue( ) As Double</b>
<b>cs</b>	double <b>get_lowestValue( )</b>
<b>java</b>	double <b>get_lowestValue( )</b>
<b>py</b>	def <b>get_lowestValue( )</b>
<b>cmd</b>	<b>YCompass target 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()**  
**compass→magneticHeading()** compass→  
**get\_magneticHeading( )**

**YCompass**

Returns the magnetic heading, regardless of the configured bearing.

js    function **get\_magneticHeading( )**  
nodejs    function **get\_magneticHeading( )**  
php    function **get\_magneticHeading( )**  
cpp    double **get\_magneticHeading( )**  
m    -(double) magneticHeading  
pas    function **get\_magneticHeading( ): double**  
vb    function **get\_magneticHeading( ) As Double**  
cs    double **get\_magneticHeading( )**  
java    double **get\_magneticHeading( )**  
py    def **get\_magneticHeading( )**  
cmd    YCompass **target get\_magneticHeading**

**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>
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 online.

**Returns :**

an instance of YModule

**compass→get\_module\_async()**  
**compass→module\_async()****YCompass**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**compass→get\_recordedData()**  
**compass→recordedData()compass→**  
**get\_recordedData()**

**YCompass**

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

js	function <b>get_recordedData( startTime, endTime)</b>
node.js	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	<b>YDataSet get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
java	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
py	<b>def get_recordedData( startTime, endTime)</b>
cmd	<b>YCompass target get_recordedData startTime 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 measure, 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 measure, 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()**  
**compass→reportFrequency()** compass→  
**get\_reportFrequency( )**

**YCompass**

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

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

**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()**  
**compass→resolution()compass→**  
**get\_resolution( )**

**YCompass**

Returns the resolution of the measured values.

<b>js</b>	function <b>get_resolution( )</b>
<b>nodejs</b>	function <b>get_resolution( )</b>
<b>php</b>	function <b>get_resolution( )</b>
<b>cpp</b>	double <b>get_resolution( )</b>
<b>m</b>	-(double) resolution
<b>pas</b>	function <b>get_resolution( ): double</b>
<b>vb</b>	function <b>get_resolution( ) As Double</b>
<b>cs</b>	double <b>get_resolution( )</b>
<b>java</b>	double <b>get_resolution( )</b>
<b>py</b>	<b>def get_resolution( )</b>
<b>cmd</b>	<b>YCompass target 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()****YCompass****compass→unit()compass→get\_unit()**

Returns the measuring unit for the relative bearing.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) <b>unit</b>
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	<b>YCompass 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	<code>function get(userData( )</code>
nodejs	<code>function get(userData( )</code>
php	<code>function get(userData( )</code>
cpp	<code>void * get(userData( )</code>
m	<code>-(void*) userData</code>
pas	<code>function get(userData( ): Tobject</code>
vb	<code>function get(userData( ) As Object</code>
cs	<code>object get(userData( )</code>
java	<code>Object get(userData( )</code>
py	<code>def get(userData( )</code>

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()****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	- <b>(BOOL) 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()****YCompass**

Preloads the compass cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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	<b>YCompass * nextCompass( )</b>
m	-(YCompass*) <b>nextCompass</b>
pas	function <b>nextCompass( )</b> : TYCompass
vb	function <b>nextCompass( )</b> As YCompass
cs	<b>YCompass nextCompass( )</b>
java	<b>YCompass 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 registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php  function registerTimedReportCallback( $callback)
cpp   int registerTimedReportCallback( YCompassTimedReportCallback callback)
m     -(int) registerTimedReportCallback : (YCompassTimedReportCallback) callback
pas   function registerTimedReportCallback( callback: TYCompassTimedReportCallback): LongInt
vb    function registerTimedReportCallback( ) As Integer
cs    int registerTimedReportCallback( TimedReportCallback callback)
java  int registerTimedReportCallback( TimedReportCallback callback)
py    def registerTimedReportCallback( callback)
```

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

**Parameters :**

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

<code>js</code>	<code>function registerValueCallback( callback)</code>
<code>node.js</code>	<code>function registerValueCallback( callback)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>int registerValueCallback( YCompassValueCallback callback)</code>
<code>m</code>	<code>-(int) registerValueCallback : (YCompassValueCallback) callback</code>
<code>pas</code>	<code>function registerValueCallback( callback: TYCompassValueCallback): LongInt</code>
<code>vb</code>	<code>function registerValueCallback( ) As Integer</code>
<code>cs</code>	<code>int registerValueCallback( ValueCallback callback)</code>
<code>java</code>	<code>int registerValueCallback( UpdateCallback callback)</code>
<code>py</code>	<code>def registerValueCallback( callback)</code>

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()**  
**compass→setHighestValue()** compass→  
**set\_highestValue( )**

**YCompass**

Changes the recorded maximal value observed.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YCompass target set_highestValue newval
```

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

**YCompass**

Changes the logical name of the compass.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the compass.

**Returns :**

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

**compass→set\_lowestValue()**  
**compass→setLowestValue()** compass→  
**set\_lowestValue( )**

**YCompass**

Changes the recorded minimal value observed.

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

**YCompass**

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   YCompass 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.

**compass→set\_resolution()**  
**compass→setResolution()****compass→**  
**set\_resolution( )**

**YCompass**

Changes the resolution of the measured physical values.

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

**YCompass**

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) <b>setUserData</b> : (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');
cpp	#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()****YCurrent****yFindCurrent()yFindCurrent( )**

Retrieves a current sensor for a given identifier.

js	function <b>yFindCurrent( func)</b>
nodejs	function <b>FindCurrent( func)</b>
php	function <b>yFindCurrent( \$func)</b>
cpp	YCurrent* <b>yFindCurrent( const string&amp; func)</b>
m	YCurrent* <b>yFindCurrent( NSString* func)</b>
pas	function <b>yFindCurrent( func: string): TYCurrent</b>
vb	function <b>yFindCurrent( ByVal func As String) As YCurrent</b>
cs	YCurrent <b>FindCurrent( string func)</b>
java	YCurrent <b>FindCurrent( String func)</b>
py	def <b>FindCurrent( 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() yFirstCurrent()yFirstCurrent( )

**YCurrent**

Starts the enumeration of current sensors currently accessible.

```
js function yFirstCurrent( )
node.js function FirstCurrent( )
php function yFirstCurrent( )
cpp YCurrent* yFirstCurrent( )
m YCurrent* yFirstCurrent( )
pas function yFirstCurrent( ): TYCurrent
vb function yFirstCurrent( ) As YCurrent
cs YCurrent FirstCurrent( )
java YCurrent FirstCurrent( )
py def FirstCurrent( )
```

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)
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  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()****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)=RELAYL01-123456.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 current sensor (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**current→get\_advertisedValue()**  
**current→advertisedValue()current→**  
**get\_advertisedValue( )**

**YCurrent**

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

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

**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()`  
`current→currentRawValue()current→`  
`get_currentRawValue( )`

**YCurrent**

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

`js` `function get_currentRawValue( )`  
`nodejs` `function get_currentRawValue( )`  
`php` `function get_currentRawValue( )`  
`cpp` `double get_currentRawValue( )`  
`m` `-(double) currentRawValue`  
`pas` `function get_currentRawValue( ): double`  
`vb` `function get_currentRawValue( ) As Double`  
`cs` `double get_currentRawValue( )`  
`java` `double get_currentRawValue( )`  
`py` `def get_currentRawValue( )`  
`cmd` `YCurrent target get_currentRawValue`

**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()**  
**current→currentValue()** current→  
**get\_currentValue()**

**YCurrent**

Returns the current measure for the current.

js	function <b>get_currentValue( )</b>
node.js	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	<b>YCurrent target 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()**  
**current→errorMessage()** current→  
**get\_errorMessage( )**

**YCurrent**

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

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

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

**Returns :**

a string corresponding to the latest error message that 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 get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the current sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the current sensor (for exemple: 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()**  
**current→functionDescriptor()current→**  
**get\_functionDescriptor( )**

**YCurrent**

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

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>node.js</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	<b>String get_functionDescriptor( )</b>
<b>py</b>	<b>def 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>
node.js	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( ) As String</b>
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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the current sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**current→get\_highestValue()**  
**current→highestValue()** current→  
**get\_highestValue( )**

**YCurrent**

Returns the maximal value observed for the current.

js    function **get\_highestValue( )**  
nodejs    function **get\_highestValue( )**  
php    function **get\_highestValue( )**  
cpp    double **get\_highestValue( )**  
m    -(double) highestValue  
pas    function **get\_highestValue( )**: double  
vb    function **get\_highestValue( )** As Double  
cs    double **get\_highestValue( )**  
java    double **get\_highestValue( )**  
py    def **get\_highestValue( )**  
cmd    YCurrent target **get\_highestValue**

**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()**  
**current→logFrequency()current→**  
**get\_logFrequency( )**

**YCurrent**

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

<b>js</b>	function <b>get_logFrequency( )</b>
<b>nodejs</b>	function <b>get_logFrequency( )</b>
<b>php</b>	function <b>get_logFrequency( )</b>
<b>cpp</b>	string <b>get_logFrequency( )</b>
<b>m</b>	-(NSString*) logFrequency
<b>pas</b>	function <b>get_logFrequency( )</b> : string
<b>vb</b>	function <b>get_logFrequency( )</b> As String
<b>cs</b>	string <b>get_logFrequency( )</b>
<b>java</b>	String <b>get_logFrequency( )</b>
<b>py</b>	def <b>get_logFrequency( )</b>
<b>cmd</b>	<b>YCurrent target 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()**  
**current→logicalName()** current→  
**get\_logicalName( )**

**YCurrent**

Returns the logical name of the current sensor.

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

**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()**  
**current→lowestValue()current→**  
**get\_lowestValue( )**

**YCurrent**

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	<b>YCurrent target 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>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**current→get\_recordedData()**  
**current→recordedData()current→**  
**get\_recordedData( )**

**YCurrent**

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

js	function <b>get_recordedData( startTime, endTime)</b>
nodejs	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	YDataSet <b>get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
java	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
py	def <b>get_recordedData( startTime, endTime)</b>
cmd	<b>YCurrent target get_recordedData startTime 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 measure, 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 measure, 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()**  
**current→reportFrequency()** current→  
**get\_reportFrequency( )**

**YCurrent**

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

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

**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>
node.js	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	<b>YCurrent target 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*) <b>unit</b>
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	<b>YCurrent target 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 get(userData)
node.js	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): TObject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

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

**Returns :**

the object stored previously by the caller.

**current→isOnline()****YCurrent**

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

js	function <b>isOnline( )</b>
node.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 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()****YCurrent**

Preloads the current sensor cache with a specified validity duration.

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

**current→loadCalibrationPoints()** **current→  
loadCalibrationPoints()**

**YCurrent**

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

js	function <b>loadCalibrationPoints( rawValues, refValues )</b>
nodejs	function <b>loadCalibrationPoints( rawValues, refValues )</b>
php	function <b>loadCalibrationPoints( &amp;\$rawValues, &amp;\$refValues )</b>
cpp	int <b>loadCalibrationPoints( vector&lt;double&gt;&amp; rawValues, vector&lt;double&gt;&amp; refValues )</b>
m	- <b>(int) loadCalibrationPoints : (NSMutableArray*) rawValues : (NSMutableArray*) refValues</b>
pas	function <b>loadCalibrationPoints( var rawValues: TDoubleArray, var refValues: TDoubleArray): LongInt</b>
vb	procedure <b>loadCalibrationPoints( )</b>
cs	int <b>loadCalibrationPoints( List&lt;double&gt; rawValues, List&lt;double&gt; refValues )</b>
java	int <b>loadCalibrationPoints( ArrayList&lt;Double&gt; rawValues, ArrayList&lt;Double&gt; refValues )</b>
py	def <b>loadCalibrationPoints( rawValues, refValues )</b>
cmd	<b>YCurrent target loadCalibrationPoints rawValues refValues</b>

**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	<b>YCurrent * nextCurrent()</b>
m	<b>-(YCurrent*) nextCurrent</b>
pas	function <b>nextCurrent()</b> : TYCurrent
vb	function <b>nextCurrent()</b> As YCurrent
cs	<b>YCurrent nextCurrent()</b>
java	<b>YCurrent nextCurrent()</b>
py	<b>def 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 registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YCurrentTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YCurrentTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYCurrentTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YCurrentValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YCurrentValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYCurrentValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**current→setHighestValue()** current→  
**set\_highestValue()**

YCurrent

Changes the recorded maximal value observed pour the current.

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

**YCurrent**

Changes the datalogger recording frequency for this function.

```
js   function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php  function set_logFrequency( $newval)
cpp   int set_logFrequency( const string& newval)
m    -(int) setLogFrequency : (NSString*) newval
pas   function set_logFrequency( newval: string): integer
vb    function set_logFrequency( ByVal newval As String) As Integer
cs    int set_logFrequency( string newval)
java  int set_logFrequency( String newval)
py    def set_logFrequency( newval)
cmd   YCurrent target set_logFrequency 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 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()**  
**current→setLogicalName()** current→  
**set\_logicalName( )**

**YCurrent**

Changes the logical name of the current sensor.

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

**YCurrent**

Changes the recorded minimal value observed pour the current.

```
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 YCurrent target set_lowestValue newval
```

**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.

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

**current→set\_resolution()**  
**current→setResolution()****current→set\_resolution()**

**YCurrent**

Changes the resolution of the measured values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YCurrent target set_resolution newval
```

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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData)</b> \$data
cpp	void <b>set(userData)</b> void* <b>data</b>
m	-(void) setUserData : (void*) <b>data</b>
pas	procedure <b>set(userData)</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>
node.js	var yoctolib = require('yoctolib');
php	var YDataLogger = yoctolib.YDataLogger;
require_once('yocto_datalogger.php');	
cpp	#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.

**datalogger→get\_functionId()**

Returns the hardware identifier of the data logger, without reference to the module.

**datalogger→get\_hardwareId()**

Returns the unique hardware identifier of the data logger in the form SERIAL . FUNCTIONID.

**datalogger→get\_logicalName()**

Returns the logical name of the data logger.

**datalogger→get\_module()**

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

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

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

**datalogger→get\_recording()**

Returns the current activation state of the data logger.

**datalogger→get\_timeUTC()**

Returns the Unix timestamp for current UTC time, if known.

**datalogger→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set(userData).

**datalogger→isOnline()**

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

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

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

**datalogger→load(msValidity)**

Preloads the data logger cache with a specified validity duration.

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

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

**datalogger→nextDataLogger()**

Continues the enumeration of data loggers started using yFirstDataLogger( ).

**datalogger→registerValueCallback(callback)**

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

**datalogger→set\_autoStart(newval)**

Changes the default activation state of the data logger on power up.

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

Changes the logical name of the data logger.

**datalogger→set\_recording(newval)**

Changes the activation state of the data logger to start/stop recording data.

**datalogger→set\_timeUTC(newval)**

Changes the current UTC time reference used for recorded data.

**datalogger→set\_userData(data)**

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

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

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

## YDataLogger.FindDataLogger() yFindDataLogger()yFindDataLogger( )

## YDataLogger

Retrieves a data logger for a given identifier.

js	function <b>yFindDataLogger( func)</b>
nodejs	function <b>FindDataLogger( func)</b>
php	function <b>yFindDataLogger( \$func)</b>
cpp	<b>YDataLogger*</b> <b>yFindDataLogger( string func)</b>
m	<b>+ (YDataLogger*) yFindDataLogger : (NSString*) func</b>
pas	function <b>yFindDataLogger( func: string): TYDataLogger</b>
vb	function <b>yFindDataLogger( ByVal func As String) As YDataLogger</b>
cs	<b>YDataLogger FindDataLogger( string func)</b>
java	<b>YDataLogger FindDataLogger( String func)</b>
py	<b>def FindDataLogger( 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()yFirstDataLogger( )**

Starts the enumeration of data loggers currently accessible.

js	function <b>yFirstDataLogger( )</b>
node.js	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()****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)=RELAYL01-123456.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 data logger (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**datalogger→forgetAllDataStreams()**  
**datalogger→forgetAllDataStreams( )****YDataLogger**

Clears the data logger memory and discards all recorded data streams.

js	function <b>forgetAllDataStreams( )</b>
node.js	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	<b>YDataLogger target 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()**  
**datalogger→advertisedValue()datalogger→**  
**get\_advertisedValue( )**

**YDataLogger**

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

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	YDataLogger <b>target 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()**  
**datalogger→autoStart()****datalogger→get\_autoStart( )**

**YDataLogger**

Returns the default activation state of the data logger on power up.

```
js function get_autoStart( )
nodejs function get_autoStart( )
php function get_autoStart( )
cpp Y_AUTOSTART_enum get_autoStart( )
m -(Y_AUTOSTART_enum) autoStart
pas function get_autoStart( ): Integer
vb function get_autoStart( ) As Integer
cs int get_autoStart( )
java int get_autoStart( )
py def get_autoStart( )
cmd YDataLogger target get_autoStart
```

**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()**  
**datalogger→currentRunIndex()datalogger→**  
**get\_currentRunIndex( )**

**YDataLogger**

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 get_currentRunIndex( )
nodejs function get_currentRunIndex( )
php function get_currentRunIndex( )
cpp int get_currentRunIndex( )
m -(int) currentRunIndex
pas function get_currentRunIndex( ): LongInt
vb function get_currentRunIndex( ) As Integer
cs int get_currentRunIndex( )
java int get_currentRunIndex( )
py def get_currentRunIndex( )
cmd YDataLogger target get_currentRunIndex
```

**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()**  
**datalogger→dataSets()datalogger→**  
**get\_dataSets( )**

**YDataLogger**

Returns a list of YDataSet objects that can be used to retrieve all measures stored by the data logger.

```
js function get_dataSets( )
nodejs function get_dataSets( )
php function get_dataSets( )
cpp vector<YDataSet> get_dataSets( )
m -(NSMutableArray*) dataSets
pas function get_dataSets( ): TYDataSetArray
vb function get_dataSets( ) As List
cs List<YDataSet> get_dataSets( )
java ArrayList<YDataSet> get_dataSets( )
py def get_dataSets( )
cmd YDataLogger target get_dataSets
```

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( v)</b>
node.js	function <b>get_dataStreams( v)</b>
php	function <b>get_dataStreams( &amp;\$v)</b>
cpp	int <b>get_dataStreams( )</b>
m	-(int) <b>dataStreams : (NSArray**) v</b>
pas	function <b>get_dataStreams( v: Tlist): integer</b>
vb	procedure <b>get_dataStreams( ByVal v As List)</b>
cs	int <b>get_dataStreams( List&lt;YDataStream&gt; v)</b>
java	int <b>get_dataStreams( ArrayList&lt;YDataStream&gt; v)</b>
py	def <b>get_dataStreams( 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()**  
**datalogger→errorMessage()****datalogger→get\_errorMessage( )**

**YDataLogger**

Returns the error message of the latest error with the data logger.

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

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

**Returns :**

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

**datalogger→get\_errorType()**  
**datalogger→errorType()datalogger→**  
**get\_errorType( )**

**YDataLogger**

Returns the numerical error code of the latest error with the data logger.

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

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the data logger object

**datalogger→get\_friendlyName()**  
**datalogger→friendlyName()****datalogger→get\_friendlyName( )**

**YDataLogger**

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

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

The returned string uses the logical names of the module and of the 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>
node.js	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	<b>String get_functionDescriptor( )</b>
py	<b>def 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()**  
**datalogger→functionId()** **datalogger→get\_functionId( )**

**YDataLogger**

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>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the data logger (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**datalogger→get\_logicalName()**  
**datalogger→logicalName()****datalogger→get\_logicalName( )**

**YDataLogger**

Returns the logical name of the data logger.

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

**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>
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 online.

**Returns :**

an instance of YModule

**datalogger→get\_module\_async()  
datalogger→module\_async()****YDataLogger**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**datalogger→get\_recording()**  
**datalogger→recording()datalogger→**  
**get\_recording( )**

**YDataLogger**

Returns the current activation state of the data logger.

<b>js</b>	function <b>get_recording( )</b>
<b>nodejs</b>	function <b>get_recording( )</b>
<b>php</b>	function <b>get_recording( )</b>
<b>cpp</b>	Y_RECORDING_enum <b>get_recording( )</b>
<b>m</b>	-(Y_RECORDING_enum) recording
<b>pas</b>	function <b>get_recording( )</b> : Integer
<b>vb</b>	function <b>get_recording( )</b> As Integer
<b>cs</b>	int <b>get_recording( )</b>
<b>java</b>	int <b>get_recording( )</b>
<b>py</b>	def <b>get_recording( )</b>
<b>cmd</b>	YDataLogger <b>target 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()**  
**datalogger→timeUTC()****datalogger→get\_timeUTC( )**

**YDataLogger**

Returns the Unix timestamp for current UTC time, if known.

**js** function **get\_timeUTC( )**  
**nodejs** function **get\_timeUTC( )**  
**php** function **get\_timeUTC( )**  
**cpp** s64 **get\_timeUTC( )**  
**m** -(s64) **timeUTC**  
**pas** function **get\_timeUTC( )**: int64  
**vb** function **get\_timeUTC( )** As Long  
**cs** long **get\_timeUTC( )**  
**java** long **get\_timeUTC( )**  
**py** def **get\_timeUTC( )**  
**cmd** YDataLogger **target get\_timeUTC**

**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)**  
**datalogger→userData()dataloader→**  
**get(userData( )**

**YDataLogger**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	function <b>get(userData( )</b>
node.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.

**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	- <b>(BOOL) 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()****YDataLogger**

Preloads the data logger cache with a specified validity duration.

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

Continues the enumeration of data loggers started using `yFirstDataLogger()`.

<code>js</code>	<code>function nextDataLogger( )</code>
<code>node.js</code>	<code>function nextDataLogger( )</code>
<code>php</code>	<code>function nextDataLogger( )</code>
<code>cpp</code>	<code>YDataLogger * nextDataLogger( )</code>
<code>m</code>	<code>-(YDataLogger*) nextDataLogger</code>
<code>pas</code>	<code>function nextDataLogger( ): TYDataLogger</code>
<code>vb</code>	<code>function nextDataLogger( ) As YDataLogger</code>
<code>cs</code>	<code>YDataLogger nextDataLogger( )</code>
<code>java</code>	<code>YDataLogger nextDataLogger( )</code>
<code>py</code>	<code>def nextDataLogger( )</code>

**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.

<code>js</code>	<code>function registerValueCallback( callback)</code>
<code>node.js</code>	<code>function registerValueCallback( callback)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>int registerValueCallback( YDataLoggerValueCallback callback)</code>
<code>m</code>	<code>-(int) registerValueCallback : (YDataLoggerValueCallback) callback</code>
<code>pas</code>	<code>function registerValueCallback( callback: TYDataLoggerValueCallback): LongInt</code>
<code>vb</code>	<code>function registerValueCallback( ) As Integer</code>
<code>cs</code>	<code>int registerValueCallback( ValueCallback callback)</code>
<code>java</code>	<code>int registerValueCallback( UpdateCallback callback)</code>
<code>py</code>	<code>def registerValueCallback( callback)</code>

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()**  
**datalogger→setAutoStart()****datalogger→set\_autoStart( )**

**YDataLogger**

Changes the default activation state of the data logger on power up.

```
js function set_autoStart( newval)
nodejs function set_autoStart( newval)
php function set_autoStart( $newval)
cpp int set_autoStart( Y_AUTOSTART_enum newval)
m -(int) setAutoStart : (Y_AUTOSTART_enum) newval
pas function set_autoStart( newval: Integer): integer
vb function set_autoStart( ByVal newval As Integer) As Integer
cs int set_autoStart( int newval)
java int set_autoStart( int newval)
py def set_autoStart( newval)
cmd YDataLogger target set_autoStart newval
```

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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YDataLogger 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 data logger.

**Returns :**

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

**datalogger→set\_recording()**  
**datalogger→setRecording()datalogger→**  
**set\_recording( )**

**YDataLogger**

Changes the activation state of the data logger to start/stop recording data.

```
js function set_recording( newval)
nodejs function set_recording( newval)
php function set_recording( $newval)
cpp int set_recording( Y_RECORDING_enum newval)
m -(int) setRecording : (Y_RECORDING_enum) newval
pas function set_recording( newval: Integer): integer
vb function set_recording( ByVal newval As Integer) As Integer
cs int set_recording( int newval)
java int set_recording( int newval)
py def set_recording( newval)
cmd YDataLogger target set_recording newval
```

**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()**  
**datalogger→setTimeUTC()****datalogger→set\_timeUTC()**

**YDataLogger**

Changes the current UTC time reference used for recorded data.

<b>js</b>	function <b>set_timeUTC( newval)</b>
<b>node.js</b>	function <b>set_timeUTC( newval)</b>
<b>php</b>	function <b>set_timeUTC( \$newval)</b>
<b>cpp</b>	int <b>set_timeUTC( s64 newval)</b>
<b>m</b>	-(int) setTimeUTC : (s64) <b>newval</b>
<b>pas</b>	function <b>set_timeUTC( newval: int64): integer</b>
<b>vb</b>	function <b>set_timeUTC( ByVal newval As Long) As Integer</b>
<b>cs</b>	int <b>set_timeUTC( long newval)</b>
<b>java</b>	int <b>set_timeUTC( long newval)</b>
<b>py</b>	def <b>set_timeUTC( newval)</b>
<b>cmd</b>	<b>YDataLogger 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)**  
**datalogger→setUserData()****datalogger→set(userData)**

**YDataLogger**

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

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

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

**Parameters :**

**data** any kind of object to be stored

## 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');
cpp #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

#### **datarun→get\_averageValue(measureName, pos)**

Returns the average value of the measure observed at the specified time period.

#### **datarun→get\_duration()**

Returns the duration (in seconds) of the data run.

#### **datarun→get\_maxValue(measureName, pos)**

Returns the maximal value of the measure observed at the specified time period.

#### **datarun→get\_measureNames()**

Returns the names of the measures recorded by the data logger.

#### **datarun→get\_minValue(measureName, pos)**

Returns the minimal value of the measure observed at the specified time period.

#### **datarun→get\_startTimeUTC()**

Returns the start time of the data run, relative to the Jan 1, 1970.

#### **datarun→get\_valueCount()**

Returns the number of values accessible in this run, given the selected data samples interval.

#### **datarun→get\_valueInterval()**

Returns the number of seconds covered by each value in this run.

#### **datarun→set\_valueInterval(valueInterval)**

Changes the number of seconds covered by each value in this run.

**datarun→get\_averageValue()**  
**datarun→averageValue()****YDataRun**

Returns the average value of the measure observed at the specified time period.

```
js function get_averageValue( measureName, pos)
nodejs function get_averageValue( measureName, pos)
php function get_averageValue( $measureName, $pos)
java double get_averageValue( String measureName, int pos)
py def get_averageValue( measureName, pos)
```

**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()**  
**datarun→duration()****YDataRun**

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()**  
**datarun→maxValue()****YDataRun**

Returns the maximal value of the measure observed at the specified time period.

```
js function get_maxValue( measureName, pos)
nodejs function get_maxValue( measureName, pos)
php function get_maxValue( $measureName, $pos)
java double get_maxValue( String measureName, int pos)
py def get_maxValue( measureName, pos)
```

**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()**  
**datarun→measureNames()****YDataRun**

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()**  
**datarun→minValue()****YDataRun**

Returns the minimal value of the measure observed at the specified time period.

```
js function get_minValue( measureName, pos)
nodejs function get_minValue( measureName, pos)
php function get_minValue( $measureName, $pos)
java double get_minValue( String measureName, int pos)
py def get_minValue( measureName, pos)
```

**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()**  
**datarun→startTimeUTC()**

---

**YDataRun**

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()**  
**datarun→valueCount()****YDataRun**

Returns the number of values accessible in this run, given the selected data samples interval.

```
js function get_valueCount( )  
nodejs function get_valueCount( )  
php function get_valueCount( )  
java int get_valueCount( )  
py def get_valueCount( )
```

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()  
datarun→valueInterval()****YDataRun**

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()  
datarun→setValueInterval()****YDataRun**

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 instanciated 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 <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 *

```

### 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()  
dataset→endTimeUTC()dataset→  
get\_endTimeUTC( )

**YDataSet**

Returns the end time of the dataset, relative to the Jan 1, 1970.

js	function get_endTimeUTC( )
nodejs	function get_endTimeUTC( )
php	function get_endTimeUTC( )
cpp	s64 get_endTimeUTC( )
m	-(s64) endTimeUTC
pas	function get_endTimeUTC( ): int64
vb	function get_endTimeUTC( ) As Long
cs	long get_endTimeUTC( )
java	long get_endTimeUTC( )
py	def get_endTimeUTC( )

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	<b>def 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>
node.js	function <b>get_measures( )</b>
php	function <b>get_measures( )</b>
cpp	vector<YMeasure> <b>get_measures( )</b>
m	-(NSMutableArray*) <b>get_measures( )</b>
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 get_preview( )
nodejs function get_preview( )
php function get_preview( )
cpp vector<YMeasure> get_preview( )
m -(NSMutableArray*) preview
pas function get_preview( ): TYMeasureArray
vb function get_preview( ) As List
cs List<YMeasure> get_preview( )
java ArrayList<YMeasure> get_preview( )
py def get_preview( )
```

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	<b>int get_progress( )</b>
m	<b>-(int) progress</b>
pas	function <b>get_progress( ): LongInt</b>
vb	function <b>get_progress( ) As Integer</b>
cs	<b>int get_progress( )</b>
java	<b>int get_progress( )</b>
py	<b>def get_progress( )</b>

When the object is instanciated 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→getStartTimeUTC()**  
**dataset→startTimeUTC()****dataset→getStartTimeUTC( )**

**YDataSet**

Returns the start time of the dataset, relative to the Jan 1, 1970.

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

In 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 **get\_unit( )**

**node.js** function **get\_unit( )**

**php** function **get\_unit( )**

**cpp** string **get\_unit( )**

**m** -(NSString\*) **unit**

**pas** function **get\_unit( )**: string

**vb** function **get\_unit( )** As String

**cs** string **get\_unit( )**

**java** String **get\_unit( )**

**py** def **get\_unit( )**

**Returns :**

a string that represents a physical unit.

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

**dataset→loadMore()****YDataSet**

Loads the the next block of measures from the dataLogger, and updates the progress indicator.

js	function <b>loadMore( )</b>
node.js	function <b>loadMore( )</b>
php	function <b>loadMore( )</b>
cpp	int <b>loadMore( )</b>
m	- <b>(int) 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	<b>def 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');
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 *

```

### YDataStream methods

#### `datastream→get_averageValue()`

Returns the average of all measures observed within this stream.

#### `datastream→get_columnCount()`

Returns the number of data columns present in this stream.

#### `datastream→get_columnNames()`

Returns the title (or meaning) of each data column present in this stream.

#### `datastream→get_data(row, col)`

Returns a single measure from the data stream, specified by its row and column index.

#### `datastream→get_dataRows()`

Returns the whole data set contained in the stream, as a bidimensional table of numbers.

#### `datastream→get_dataSamplesIntervalMs()`

Returns the number of milliseconds between two consecutive rows of this data stream.

#### `datastream→get_duration()`

Returns the approximate duration of this stream, in seconds.

#### `datastream→get_maxValue()`

Returns the largest measure observed within this stream.

#### `datastream→get_minValue()`

Returns the smallest measure observed within this stream.

#### `datastream→getRowCount()`

Returns the number of data rows present in this stream.

#### `datastream→get_runIndex()`

Returns the run index of the data stream.

#### `datastream→get_startTime()`

Returns the relative start time of the data stream, measured in seconds.

#### `datastream→get_startTimeUTC()`

### **3. Reference**

---

Returns the start time of the data stream, relative to the Jan 1, 1970.

**datastream→get\_averageValue()**  
**datastream→averageValue()datastream→**  
**get\_averageValue()**

**YDataStream**

Returns the average of all measures observed within this stream.

js	function <b>get_averageValue( )</b>
node.js	function <b>get_averageValue( )</b>
php	function <b>get_averageValue( )</b>
cpp	double <b>get_averageValue( )</b>
m	-(double) <b>averageValue</b>
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	<b>def get_averageValue( )</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 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()**  
**datastream→columnCount()****datastream→get\_columnCount( )**

**YDataStream**

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()**  
**datastream→columnNames()** **datastream→get\_columnNames( )**

**YDataStream**

Returns the title (or meaning) of each data column present in this stream.

js	function <b>get_columnNames( )</b>
node.js	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> ( \$row, \$col)
cpp	<b>double get_data( int row, int col)</b>
m	- <b>(double) data : (int) row</b> <b>: (int) 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	<b>double get_data( int row, int col)</b>
java	<b>double get_data( int row, int col)</b>
py	<b>def get_data( row, 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()**  
**datastream→dataRows()datastream→**  
**get\_dataRows( )**

**YDataStream**

Returns the whole data set contained in the stream, as a bidimensional table of numbers.

js	function <b>get_dataRows( )</b>
node.js	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> : TDoubleArrayList
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()**  
**datastream→dataSamplesIntervalMs()datastream→**  
**get\_dataSamplesIntervalMs( )**

**YDataStream**

Returns the number of milliseconds between two consecutive rows of this data stream.

js	function get_dataSamplesIntervalMs( )
nodejs	function get_dataSamplesIntervalMs( )
php	function get_dataSamplesIntervalMs( )
cpp	int get_dataSamplesIntervalMs( )
m	-(int) dataSamplesIntervalMs
pas	function get_dataSamplesIntervalMs( ): LongInt
vb	function get_dataSamplesIntervalMs( ) As Integer
cs	int get_dataSamplesIntervalMs( )
java	int get_dataSamplesIntervalMs( )
py	def get_dataSamplesIntervalMs( )

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()**  
**datastream→duration()datastream→get\_duration( )**

**YDataStream**

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) <b>duration</b>
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	<b>def 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()`  
`datastream→maxValue()``datastream→get_maxValue( )`

**YDataStream**

Returns the largest measure observed within this stream.

`js` `function get_maxValue( )`  
`nodejs` `function get_maxValue( )`  
`php` `function get_maxValue( )`  
`cpp` `double get_maxValue( )`  
`m` `-(double) maxValue`  
`pas` `function get_maxValue( ): double`  
`vb` `function get_maxValue( ) As Double`  
`cs` `double get_maxValue( )`  
`java` `double get_maxValue( )`  
`py` `def get_maxValue( )`

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()**  
**datastream→minValue()** **datastream→get\_minValue( )**

**YDataStream**

Returns the smallest measure observed within this stream.

js	function <b>get_minValue( )</b>
node.js	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	<b>def 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→getRowCount()**  
**datastream→rowCount()** **datastream→getRowCount()**

**YDataStream**

Returns the number of data rows present in this stream.

**js** `function getRowCount( )`  
**nodejs** `function getRowCount( )`  
**php** `function getRowCount( )`  
**cpp** `int getCount( )`  
**m** `-(int) rowCount`  
**pas** `function getCount( ): LongInt`  
**vb** `function getCount( ) As Integer`  
**cs** `int getCount( )`  
**java** `int getCount( )`  
**py** `def getCount( )`

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()**  
**datastream→runIndex()datastream→**  
**get\_runIndex( )**

**YDataStream**

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) <b>runIndex</b>
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→getStartTime()**  
**datastream→startTime()datastream→**  
**getStartTime( )**

**YDataStream**

Returns the relative start time of the data stream, measured in seconds.

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

**Returns :**

an unsigned number corresponding to the number of seconds between the start of the run and the beginning of this data stream.

**datastream→getStartTimeUTC()**  
**datastream→startTimeUTC()datastream→**  
**getStartTimeUTC()**

**YDataStream**

Returns the start time of the data stream, relative to the Jan 1, 1970.

js	function <b>getStartTimeUTC( )</b>
node.js	function <b>getStartTimeUTC( )</b>
php	function <b>getStartTimeUTC( )</b>
cpp	s64 <b>getStartTimeUTC( )</b>
m	-(s64) startTimeUTC
pas	function <b>getStartTimeUTC( )</b> : int64
vb	function <b>getStartTimeUTC( )</b> As Long
cs	long <b>getStartTimeUTC( )</b>
java	long <b>getStartTimeUTC( )</b>
py	<b>def getStartTimeUTC( )</b>

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');
cpp #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)**

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()****YDigitalIO****yFindDigitalIO()yFindDigitalIO( )**

Retrieves a digital IO port for a given identifier.

<code>js</code>	<code>function yFindDigitalIO( func)</code>
<code>node.js</code>	<code>function FindDigitalIO( func)</code>
<code>php</code>	<code>function yFindDigitalIO( \$func)</code>
<code>cpp</code>	<code>YDigitalIO* yFindDigitalIO( const string&amp; func)</code>
<code>m</code>	<code>YDigitalIO* yFindDigitalIO( NSString* func)</code>
<code>pas</code>	<code>function yFindDigitalIO( func: string): TYDigitalIO</code>
<code>vb</code>	<code>function yFindDigitalIO( ByVal func As String) As YDigitalIO</code>
<code>cs</code>	<code>YDigitalIO FindDigitalIO( string func)</code>
<code>java</code>	<code>YDigitalIO FindDigitalIO( String func)</code>
<code>py</code>	<code>def FindDigitalIO( 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 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()****YDigitalIO****yFirstDigitalIO()yFirstDigitalIO( )**

Starts the enumeration of digital IO ports currently accessible.

js	function <b>yFirstDigitalIO( )</b>
node.js	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.

<b>js</b>	function <b>delayedPulse( bitno, ms_delay, ms_duration)</b>
<b>nodejs</b>	function <b>delayedPulse( bitno, ms_delay, ms_duration)</b>
<b>php</b>	function <b>delayedPulse( \$bitno, \$ms_delay, \$ms_duration)</b>
<b>cpp</b>	int <b>delayedPulse( int bitno, int ms_delay, int ms_duration)</b>
<b>m</b>	- <b>(int) delayedPulse : (int) bitno : (int) ms_delay : (int) ms_duration</b>
<b>pas</b>	function <b>delayedPulse( bitno: LongInt,</b> <b>                  ms_delay: LongInt,</b> <b>                  ms_duration: LongInt): LongInt</b>
<b>vb</b>	function <b>delayedPulse( ) As Integer</b>
<b>cs</b>	int <b>delayedPulse( int bitno, int ms_delay, int ms_duration)</b>
<b>java</b>	int <b>delayedPulse( int bitno, int ms_delay, int ms_duration)</b>
<b>py</b>	def <b>delayedPulse( bitno, ms_delay, ms_duration)</b>
<b>cmd</b>	<b>YDigitalIO target delayedPulse bitno ms_delay 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 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)=RELAYL01-123456.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 digital IO port (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**digitalio→get\_advertisedValue()**  
**digitalio→advertisedValue()****digitalio→get\_advertisedValue( )**

**YDigitalIO**

Returns the current value of the digital IO port (no more than 6 characters).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YDigitalIO target 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()**  
**digitalio→bitDirection()****digitalio→get\_bitDirection()****YDigitalIO**

Returns the direction of a single bit from the I/O port (0 means the bit is an input, 1 an output).

js	function get_bitDirection( bitno)
nodejs	function get_bitDirection( bitno)
php	function get_bitDirection( \$bitno)
cpp	int get_bitDirection( int bitno)
m	-(int) bitDirection : (int) bitno
pas	function get_bitDirection( bitno: LongInt): LongInt
vb	function get_bitDirection( ) As Integer
cs	int get_bitDirection( int bitno)
java	int get_bitDirection( int bitno)
py	def get_bitDirection( bitno)
cmd	YDigitalIO target get_bitDirection bitno

**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()**  
**digitalio→bitOpenDrain()digitalio→**  
**get\_bitOpenDrain()**

**YDigitalIO**

Returns the type of electrical interface of a single bit from the I/O port.

js	function <b>get_bitOpenDrain( bitno)</b>
nodejs	function <b>get_bitOpenDrain( bitno)</b>
php	function <b>get_bitOpenDrain( \$bitno)</b>
cpp	int <b>get_bitOpenDrain( int bitno)</b>
m	-(int) <b>bitOpenDrain : (int) bitno</b>
pas	function <b>get_bitOpenDrain( bitno: LongInt): LongInt</b>
vb	function <b>get_bitOpenDrain( ) As Integer</b>
cs	int <b>get_bitOpenDrain( int bitno)</b>
java	int <b>get_bitOpenDrain( int bitno)</b>
py	def <b>get_bitOpenDrain( 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()**  
**digitalio→bitPolarity()****digitalio→get\_bitPolarity()**

**YDigitalIO**

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 get_bitPolarity( bitno)
nodejs function get_bitPolarity( bitno)
php function get_bitPolarity( $bitno)
cpp int get_bitPolarity( int bitno)
m -(int) bitPolarity : (int) bitno
pas function get_bitPolarity( bitno: LongInt): LongInt
vb function get_bitPolarity( ) As Integer
cs int get_bitPolarity( int bitno)
java int get_bitPolarity( int bitno)
py def get_bitPolarity( bitno)
cmd YDigitalIO target get_bitPolarity bitno
```

**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( bitno)</b>
node.js	function <b>get_bitState( bitno)</b>
php	function <b>get_bitState( \$bitno)</b>
cpp	int <b>get_bitState( int bitno)</b>
m	-(int) bitState : (int) <b>bitno</b>
pas	function <b>get_bitState( bitno: LongInt): LongInt</b>
vb	function <b>get_bitState( ) As Integer</b>
cs	int <b>get_bitState( int bitno)</b>
java	int <b>get_bitState( int bitno)</b>
py	def <b>get_bitState( bitno)</b>
cmd	<b>YDigitalIO target get_bitState 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→getErrorMessage()**  
**digitalio→errorMessage()****digitalio→getErrorMessage( )**

**YDigitalIO**

Returns the error message of the latest error with the digital IO port.

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

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()**  
**digitalio→errorType()digitalio→**  
**get\_errorType( )**

**YDigitalIO**

Returns the numerical error code of the latest error with the digital IO port.

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

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the digital IO port object

**digitalio→get\_friendlyName()**  
**digitalio→friendlyName()****digitalio→get\_friendlyName( )**

**YDigitalIO**

Returns a global identifier of the digital IO port in the format MODULE\_NAME . FUNCTION\_NAME.

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

The returned string uses the logical names of the module and of the 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()**  
**digitalio→functionDescriptor()digitalio→**  
**get\_functionDescriptor( )**

**YDigitalIO**

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

js	function <b>get_functionDescriptor( )</b>
node.js	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.

**digitalio→get\_functionId()**  
**digitalio→functionId()** digitalio→  
**get\_functionId( )**

**YDigitalIO**

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()**  
**digitalio→hardwareId()** digitalio→  
**get.hardwareId( )**

**YDigitalIO**

Returns the unique hardware identifier of the digital IO port in the form SERIAL.FUNCTIONID.

js	function <b>get.hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the digital IO port (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**digitalio→get\_logicalName()**  
**digitalio→logicalName()****digitalio→get\_logicalName( )**

**YDigitalIO**

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	<b>YDigitalIO target 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> : 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 online.

**Returns :**

an instance of YModule

## digitalio→get\_module\_async() digitalio→module\_async()

YDigitalIO

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

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

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

### Returns :

nothing : the result is provided to the callback.

**digitalio→get\_outputVoltage()**  
**digitalio→outputVoltage()****digitalio→get\_outputVoltage( )**

**YDigitalIO**

Returns the voltage source used to drive output bits.

<b>js</b>	function <b>get_outputVoltage( )</b>
<b>nodejs</b>	function <b>get_outputVoltage( )</b>
<b>php</b>	function <b>get_outputVoltage( )</b>
<b>cpp</b>	Y_OUTPUTVOLTAGE_enum <b>get_outputVoltage( )</b>
<b>m</b>	-(Y_OUTPUTVOLTAGE_enum) outputVoltage
<b>pas</b>	function <b>get_outputVoltage( )</b> : Integer
<b>vb</b>	function <b>get_outputVoltage( )</b> As Integer
<b>cs</b>	int <b>get_outputVoltage( )</b>
<b>java</b>	int <b>get_outputVoltage( )</b>
<b>py</b>	def <b>get_outputVoltage( )</b>
<b>cmd</b>	<b>YDigitalIO target 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()**  
**digitalio→portDirection()digitalio→**  
**get\_portDirection( )**

**YDigitalIO**

Returns the IO direction of all bits of the port: 0 makes a bit an input, 1 makes it an output.

js function **get\_portDirection( )**  
nodejs function **get\_portDirection( )**  
php function **get\_portDirection( )**  
cpp int **get\_portDirection( )**  
m -(int) **portDirection**  
pas function **get\_portDirection( )**: LongInt  
vb function **get\_portDirection( )** As Integer  
cs int **get\_portDirection( )**  
java int **get\_portDirection( )**  
py def **get\_portDirection( )**  
cmd YDigitalIO target **get\_portDirection**

**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()**  
**digitalio→portOpenDrain()****digitalio→get\_portOpenDrain()**

**YDigitalIO**

Returns the electrical interface for each bit of the port.

<b>js</b>	function <b>get_portOpenDrain( )</b>
<b>nodejs</b>	function <b>get_portOpenDrain( )</b>
<b>php</b>	function <b>get_portOpenDrain( )</b>
<b>cpp</b>	int <b>get_portOpenDrain( )</b>
<b>m</b>	-(int) <b>portOpenDrain</b>
<b>pas</b>	function <b>get_portOpenDrain( )</b> : LongInt
<b>vb</b>	function <b>get_portOpenDrain( )</b> As Integer
<b>cs</b>	int <b>get_portOpenDrain( )</b>
<b>java</b>	int <b>get_portOpenDrain( )</b>
<b>py</b>	def <b>get_portOpenDrain( )</b>
<b>cmd</b>	<b>YDigitalIO target 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\_PORTOPENRAIN\_INVALID**.

**digitalio→get\_portPolarity()**  
**digitalio→portPolarity()** digitalio→  
**get\_portPolarity()**

**YDigitalIO**

Returns the polarity of all the bits of the port.

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

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) <b>portSize</b>
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	<b>YDigitalIO target 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\_PORTSIZERO\_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>
node.js	function <b>get_portState( )</b>
php	function <b>get_portState( )</b>
cpp	int <b>get_portState( )</b>
m	-(int) <b>portState</b>
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	<b>YDigitalIO target 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	<code>function get(userData) </code>
nodejs	<code>function get(userData) </code>
php	<code>function get(userData) </code>
cpp	<code>void * get(userData) </code>
m	<code>-(void*) userData </code>
pas	<code>function get(userData): Tobject </code>
vb	<code>function get(userData) As Object </code>
cs	<code>object get(userData) </code>
java	<code>Object get(userData) </code>
py	<code>def get(userData) </code>

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.

**digitalio→isOnline()digitalio→isOnline( )****YDigitalIO**

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	- <b>(BOOL) 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	<b>def 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.

**digitalio→load()****YDigitalIO**

Preloads the digital IO port cache with a specified validity duration.

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

<code>js</code>	<code>function nextDigitalIO( )</code>
<code>node.js</code>	<code>function nextDigitalIO( )</code>
<code>php</code>	<code>function nextDigitalIO( )</code>
<code>cpp</code>	<code>YDigitalIO * nextDigitalIO( )</code>
<code>m</code>	<code>-(YDigitalIO*) nextDigitalIO</code>
<code>pas</code>	<code>function nextDigitalIO( ): TYDigitalIO</code>
<code>vb</code>	<code>function nextDigitalIO( ) As YDigitalIO</code>
<code>cs</code>	<code>YDigitalIO nextDigitalIO( )</code>
<code>java</code>	<code>YDigitalIO nextDigitalIO( )</code>
<code>py</code>	<code>def nextDigitalIO( )</code>

**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.

<b>js</b>	<code>function pulse( bitno, ms_duration)</code>
<b>nodejs</b>	<code>function pulse( bitno, ms_duration)</code>
<b>php</b>	<code>function pulse( \$bitno, \$ms_duration)</code>
<b>cpp</b>	<code>int pulse( int bitno, int ms_duration)</code>
<b>m</b>	<code>-(int) pulse : (int) bitno : (int) ms_duration</code>
<b>pas</b>	<code>function pulse( bitno: LongInt, ms_duration: LongInt): LongInt</code>
<b>vb</b>	<code>function pulse( ) As Integer</code>
<b>cs</b>	<code>int pulse( int bitno, int ms_duration)</code>
<b>java</b>	<code>int pulse( int bitno, int ms_duration)</code>
<b>py</b>	<code>def pulse( bitno, ms_duration)</code>
<b>cmd</b>	<code>YDigitalIO target pulse bitno ms_duration</code>

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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YDigitalIOValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YDigitalIOValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYDigitalIOValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**digitalio→setBitDirection()digitalio→**  
**set\_bitDirection()**

YDigitalIO

Changes the direction of a single bit from the I/O port.

js	function <b>set_bitDirection( bitno, bitdirection)</b>
nodejs	function <b>set_bitDirection( bitno, bitdirection)</b>
php	function <b>set_bitDirection( \$bitno, \$bitdirection)</b>
cpp	int <b>set_bitDirection( int bitno, int bitdirection)</b>
m	-(int) setBitDirection : (int) <b>bitno</b> : (int) <b>bitdirection</b>
pas	function <b>set_bitDirection( bitno: LongInt, bitdirection: LongInt): LongInt</b>
vb	function <b>set_bitDirection( ) As Integer</b>
cs	int <b>set_bitDirection( int bitno, int bitdirection)</b>
java	int <b>set_bitDirection( int bitno, int bitdirection)</b>
py	def <b>set_bitDirection( bitno, bitdirection)</b>
cmd	YDigitalIO target <b>set_bitDirection bitno 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( )**

**YDigitalIO**

Changes the electrical interface of a single bit from the I/O port.

```
js function set_bitOpenDrain( bitno, opendrain)
nodejs function set_bitOpenDrain( bitno, opendrain)
php function set_bitOpenDrain( $bitno, $opendrain)
cpp int set_bitOpenDrain( int bitno, int opendrain)
m -(int) setBitOpenDrain : (int) bitno : (int) opendrain
pas function set_bitOpenDrain( bitno: LongInt, opendrain: LongInt): LongInt
vb function set_bitOpenDrain( ) As Integer
cs int set_bitOpenDrain( int bitno, int opendrain)
java int set_bitOpenDrain( int bitno, int opendrain)
py def set_bitOpenDrain( bitno, opendrain)
cmd YDigitalIO target set_bitOpenDrain bitno opendrain
```

#### 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()**  
**digitalio→setBitPolarity()digitalio→**  
**set\_bitPolarity( )**

**YDigitalIO**

Changes the polarity of a single bit from the I/O port.

js	function <b>set_bitPolarity( bitno, bitpolarity)</b>
nodejs	function <b>set_bitPolarity( bitno, bitpolarity)</b>
php	function <b>set_bitPolarity( \$bitno, \$bitpolarity)</b>
cpp	int <b>set_bitPolarity( int bitno, int bitpolarity)</b>
m	-(int) setBitPolarity : (int) <b>bitno</b> : (int) <b>bitpolarity</b>
pas	function <b>set_bitPolarity( bitno: LongInt, bitpolarity: LongInt): LongInt</b>
vb	function <b>set_bitPolarity( ) As Integer</b>
cs	int <b>set_bitPolarity( int bitno, int bitpolarity)</b>
java	int <b>set_bitPolarity( int bitno, int bitpolarity)</b>
py	def <b>set_bitPolarity( bitno, 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()**  
**digitalio→setBitState()****digitalio→set\_bitState( )**

**YDigitalIO**

Sets a single bit of the I/O port.

```
js function set_bitState( bitno, bitstate)
nodejs function set_bitState( bitno, bitstate)
php function set_bitState( $bitno, $bitstate)
cpp int set_bitState( int bitno, int bitstate)
m -(int) setBitState : (int) bitno : (int) bitstate
pas function set_bitState( bitno: LongInt, bitstate: LongInt): LongInt
vb function set_bitState( ) As Integer
cs int set_bitState( int bitno, int bitstate)
java int set_bitState( int bitno, int bitstate)
py def set_bitState( bitno, bitstate)
cmd YDigitalIO target set_bitState 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()**  
**digitalio→setLogicalName()****digitalio→**  
**set\_logicalName( )**

**YDigitalIO**

Changes the logical name of the digital IO port.

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

<b>js</b>	function <b>set_outputVoltage( newval)</b>
<b>nodejs</b>	function <b>set_outputVoltage( newval)</b>
<b>php</b>	function <b>set_outputVoltage( \$newval)</b>
<b>cpp</b>	int <b>set_outputVoltage( Y_OUTPUTVOLTAGE_enum newval)</b>
<b>m</b>	-(int) <b>setOutputVoltage : (Y_OUTPUTVOLTAGE_enum) newval</b>
<b>pas</b>	function <b>set_outputVoltage( newval: Integer): integer</b>
<b>vb</b>	function <b>set_outputVoltage( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_outputVoltage( int newval)</b>
<b>java</b>	int <b>set_outputVoltage( int newval)</b>
<b>py</b>	def <b>set_outputVoltage( newval)</b>
<b>cmd</b>	<b>YDigitalIO target set_outputVoltage 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( newval)</b>
node.js	function <b>set_portDirection( newval)</b>
php	function <b>set_portDirection( \$newval)</b>
cpp	int <b>set_portDirection( int newval)</b>
m	-(int) <b>setPortDirection : (int) newval</b>
pas	function <b>set_portDirection( newval: LongInt): integer</b>
vb	function <b>set_portDirection( ByVal newval As Integer) As Integer</b>
cs	int <b>set_portDirection( int newval)</b>
java	int <b>set_portDirection( int newval)</b>
py	def <b>set_portDirection( newval)</b>
cmd	<b>YDigitalIO target set_portDirection 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()**  
**digitalio→setPortOpenDrain()****digitalio→set\_portOpenDrain( )**

**YDigitalIO**

Changes the electrical interface for each bit of the port.

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

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()  
digitalio→setPortPolarity()digitalio→  
set\_portPolarity()**

YDigitalIO

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( newval)</b>
node.js	function <b>set_portPolarity( newval)</b>
php	function <b>set_portPolarity( \$newval)</b>
cpp	int <b>set_portPolarity( int newval)</b>
m	-(int) <b>setPortPolarity : (int) newval</b>
pas	function <b>set_portPolarity( newval: LongInt): integer</b>
vb	function <b>set_portPolarity( ByVal newval As Integer) As Integer</b>
cs	int <b>set_portPolarity( int newval)</b>
java	int <b>set_portPolarity( int newval)</b>
py	def <b>set_portPolarity( newval)</b>
cmd	<b>YDigitalIO 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()**  
**digitalio→setPortState()****digitalio→set\_portState( )**

**YDigitalIO**

Changes the digital IO port state: bit 0 represents input 0, and so on.

<b>js</b>	function <b>set_portState( newval)</b>
<b>nodejs</b>	function <b>set_portState( newval)</b>
<b>php</b>	function <b>set_portState( \$newval)</b>
<b>cpp</b>	int <b>set_portState( int newval)</b>
<b>m</b>	-(int) <b>setPortState : (int) newval</b>
<b>pas</b>	function <b>set_portState( newval: LongInt): integer</b>
<b>vb</b>	function <b>set_portState( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_portState( int newval)</b>
<b>java</b>	int <b>set_portState( int newval)</b>
<b>py</b>	def <b>set_portState( newval)</b>
<b>cmd</b>	<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)**  
**digitalio→setUserData()** digitalio→  
**set(userData)**

YDigitalIO

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

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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( bitno)</b>
node.js	function <b>toggle_bitState( bitno)</b>
php	function <b>toggle_bitState( \$bitno)</b>
cpp	int <b>toggle_bitState( int bitno)</b>
m	- <b>(int) toggle_bitState : (int) bitno</b>
pas	function <b>toggle_bitState( bitno: LongInt): LongInt</b>
vb	function <b>toggle_bitState( ) As Integer</b>
cs	int <b>toggle_bitState( int bitno)</b>
java	int <b>toggle_bitState( int bitno)</b>
py	def <b>toggle_bitState( bitno)</b>
cmd	<b>YDigitalIO 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)**

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()****YDisplay****yFindDisplay()yFindDisplay( )**

Retrieves a display for a given identifier.

js	function <b>yFindDisplay( func)</b>
nodejs	function <b>FindDisplay( func)</b>
php	function <b>yFindDisplay( \$func)</b>
cpp	YDisplay* <b>yFindDisplay( string func)</b>
m	+( <b>YDisplay*</b> ) <b>yFindDisplay : (NSString*) func</b>
pas	function <b>yFindDisplay( func: string): TYDisplay</b>
vb	function <b>yFindDisplay( ByVal func As String) As YDisplay</b>
cs	YDisplay <b>FindDisplay( string func)</b>
java	YDisplay <b>FindDisplay( String func)</b>
py	def <b>FindDisplay( 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()****YDisplay****yFirstDisplay()yFirstDisplay( )**

Starts the enumeration of displays currently accessible.

js	function <b>yFirstDisplay( )</b>
node.js	function <b>FirstDisplay( )</b>
php	function <b>yFirstDisplay( )</b>
cpp	<b>YDisplay*</b> <b>yFirstDisplay( )</b>
m	<b>YDisplay*</b> <b>yFirstDisplay( )</b>
pas	function <b>yFirstDisplay( )</b> : TYDisplay
vb	function <b>yFirstDisplay( )</b> As YDisplay
cs	<b>YDisplay FirstDisplay( )</b>
java	<b>YDisplay 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 copyLayerContent( srcLayerId, dstLayerId)
nodejs function copyLayerContent( srcLayerId, dstLayerId)
php  function copyLayerContent( $srcLayerId, $dstLayerId)
cpp   int copyLayerContent( int srcLayerId, int dstLayerId)
m     -(int) copyLayerContent : (int) srcLayerId
                  : (int) dstLayerId

pas  function copyLayerContent( srcLayerId: LongInt,
                               dstLayerId: LongInt): LongInt

vb   function copyLayerContent( ) As Integer
cs    int copyLayerContent( int srcLayerId, int dstLayerId)
java  int copyLayerContent( int srcLayerId, int dstLayerId)
py    def copyLayerContent( srcLayerId, dstLayerId)
cmd   YDisplay target copyLayerContent srcLayerId dstLayerId

```

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	- <b>(NSString*) describe</b>
pas	function <b>describe( ): string</b>
vb	function <b>describe( ) As String</b>
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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)=RELAYL01-123456.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 display (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**display→fade()display→fade( )****YDisplay**

Smoothly changes the brightness of the screen to produce a fade-in or fade-out effect.

```
js function fade( brightness, duration)
nodejs function fade( brightness, duration)
php function fade( $brightness, $duration)
cpp int fade( int brightness, int duration)
m -(int) fade : (int) brightness : (int) duration
pas function fade( brightness: LongInt, duration: LongInt): LongInt
vb function fade( ) As Integer
cs int fade( int brightness, int duration)
java int fade( int brightness, int duration)
py def fade( brightness, duration)
cmd YDisplay target fade brightness duration
```

**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()**  
**display→advertisedValue()display→**  
**get\_advertisedValue( )****YDisplay**

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

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

**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	<b>YDisplay target 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()  
display→displayHeight()display→  
get\_displayHeight( )****YDisplay**

Returns the display height, in pixels.

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

**Returns :**

an integer corresponding to the display height, in pixels

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

**display→get\_displayLayer()**  
**display→displayLayer()display→**  
**get\_displayLayer( )**

**YDisplay**

Returns a YDisplayLayer object that can be used to draw on the specified layer.

js	function <b>get_displayLayer( layerId)</b>
node.js	function <b>get_displayLayer( layerId)</b>
php	function <b>get_displayLayer( \$layerId)</b>
cpp	<b>YDisplayLayer* get_displayLayer( unsigned layerId)</b>
m	-( <b>YDisplayLayer*</b> ) <b>displayLayer</b> : ( <b>unsigned</b> ) <b>layerId</b>
vb	function <b>get_displayLayer( )</b> As <b>YDisplayLayer</b>
cs	<b>YDisplayLayer get_displayLayer( int layerId)</b>
java	synchronized <b>YDisplayLayer get_displayLayer( int layerId)</b>
py	<b>def get_displayLayer( 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 get_displayType( )
nodejs function get_displayType( )
php function get_displayType( )
cpp Y_DISPLAYTYPE_enum get_displayType( )
m -(Y_DISPLAYTYPE_enum) displayType
pas function get_displayType( ): Integer
vb function get_displayType( ) As Integer
cs int get_displayType( )
java int get_displayType( )
py def get_displayType( )
cmd YDisplay target get_displayType
```

**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()  
display→displayWidth()display→  
get\_displayWidth()****YDisplay**

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) <b>displayWidth</b>
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	<b>YDisplay target 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 get_enabled( )
node.js function get_enabled( )
php function get_enabled( )
cpp Y_ENABLED_enum get_enabled( )
m -(Y_ENABLED_enum) enabled
pas function get_enabled( ): Integer
vb function get_enabled( ) As Integer
cs int get_enabled( )
java int get_enabled( )
py def get_enabled( )
cmd YDisplay target get_enabled
```

**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→getErrorMessage()**  
**display→errorMessage()display→**  
**getErrorMessage( )**

**YDisplay**

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

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

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>
node.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 the display object

**display→get\_friendlyName()**  
**display→friendlyName()display→**  
**get\_friendlyName( )**

**YDisplay**

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

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

The returned string uses the logical names of the module and of the display if they are defined, otherwise the serial number of the module and the hardware identifier of the display (for exemple: 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()** YDisplay

**display→functionDescriptor()display→get\_functionDescriptor( )**

---

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

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the display. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the display (ex: RELAYL01-123456.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 get_layerCount( )
nodejs function get_layerCount( )
php function get_layerCount( )
cpp int get_layerCount( )
m -(int) layerCount
pas function get_layerCount( ): LongInt
vb function get_layerCount( ) As Integer
cs int get_layerCount( )
java int get_layerCount( )
py def get_layerCount( )
cmd YDisplay target get_layerCount
```

**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()**  
**display→layerHeight()** **display→get\_layerHeight( )**

**YDisplay**

Returns the height of the layers to draw on, in pixels.

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

**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 get_layerWidth( )
nodejs function get_layerWidth( )
php function get_layerWidth( )
cpp int get_layerWidth( )
m -(int) layerWidth
pas function get_layerWidth( ): LongInt
vb function get_layerWidth( ) As Integer
cs int get_layerWidth( )
java int get_layerWidth( )
py def get_layerWidth( )
cmd YDisplay target get_layerWidth
```

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

**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( ): TYModule</b>
vb	function <b>get_module( ) As YModule</b>
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 online.

**Returns :**

an instance of YModule

**display→get\_module\_async()**  
**display→module\_async()****YDisplay**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**display→get\_orientation()**  
**display→orientation()display→**  
**get\_orientation( )**

**YDisplay**

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	<b>YDisplay target 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>
node.js	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	<b>YDisplay target 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	<code>function get(userData) </code>
nodejs	<code>function get(userData) </code>
php	<code>function get(userData) </code>
cpp	<code>void * get(userData) </code>
m	<code>-(void*) userData </code>
pas	<code>function get(userData): Tobject </code>
vb	<code>function get(userData) As Object </code>
cs	<code>object get(userData) </code>
java	<code>Object get(userData) </code>
py	<code>def get(userData) </code>

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

Starts to record all display commands into a sequence, for later replay.

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

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	<b>YDisplay * nextDisplay( )</b>
m	<b>-(YDisplay*) nextDisplay</b>
pas	function <b>nextDisplay( ): TYDisplay</b>
vb	function <b>nextDisplay( ) As YDisplay</b>
cs	<b>YDisplay nextDisplay( )</b>
java	<b>YDisplay nextDisplay( )</b>
py	<b>def 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 pauseSequence( delay_ms)
node.js	function pauseSequence( delay_ms)
php	function pauseSequence( \$delay_ms)
cpp	int pauseSequence( int delay_ms)
m	-(int) pauseSequence : (int) delay_ms
pas	function pauseSequence( delay_ms: LongInt): LongInt
vb	function pauseSequence( ) As Integer
cs	int pauseSequence( int delay_ms)
java	int pauseSequence( int delay_ms)
py	def pauseSequence( delay_ms)
cmd	YDisplay target pauseSequence delay_ms

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 playSequence( sequenceName)
nodejs	function playSequence( sequenceName)
php	function playSequence( \$sequenceName)
cpp	int playSequence( string sequenceName)
m	-(int) playSequence : (NSString*) sequenceName
pas	function playSequence( sequenceName: string): LongInt
vb	function playSequence( ) As Integer
cs	int playSequence( string sequenceName)
java	int playSequence( String sequenceName)
py	def playSequence( sequenceName)
cmd	YDisplay target playSequence sequenceName

**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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YDisplayValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YDisplayValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYDisplayValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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	- <b>(int) resetAll</b>
pas	function <b>resetAll( ): LongInt</b>
vb	function <b>resetAll( ) As Integer</b>
cs	int <b>resetAll( )</b>
java	int <b>resetAll( )</b>
py	def <b>resetAll( )</b>
cmd	YDisplay <b>target 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 saveSequence( sequenceName)
nodejs function saveSequence( sequenceName)
php function saveSequence( $sequenceName)
cpp int saveSequence( string sequenceName)
m -(int) saveSequence : (NSString*) sequenceName
pas function saveSequence( sequenceName: string): LongInt
vb function saveSequence( ) As Integer
cs int saveSequence( string sequenceName)
java int saveSequence( String sequenceName)
py def saveSequence( sequenceName)
cmd YDisplay target saveSequence sequenceName
```

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()**  
**display→setBrightness()display→**  
**set\_brightness( )**

**YDisplay**

Changes the brightness of the display.

js	function <b>set_brightness( newval)</b>
nodejs	function <b>set_brightness( newval)</b>
php	function <b>set_brightness( \$newval)</b>
cpp	int <b>set_brightness( int newval)</b>
m	-(int) setBrightness : (int) <b>newval</b>
pas	function <b>set_brightness( newval: LongInt): integer</b>
vb	function <b>set_brightness( ByVal newval As Integer) As Integer</b>
cs	int <b>set_brightness( int newval)</b>
java	int <b>set_brightness( int newval)</b>
py	def <b>set_brightness( newval)</b>
cmd	<b>YDisplay target set_brightness 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 set_enabled( newval)
node.js function set_enabled( newval)
php function set_enabled( $newval)
cpp int set_enabled( Y_ENABLED_enum newval)
m -(int) setEnabled : (Y_ENABLED_enum) newval
pas function set_enabled( newval: Integer): integer
vb function set_enabled( ByVal newval As Integer) As Integer
cs int set_enabled( int newval)
java int set_enabled( int newval)
py def set_enabled( newval)
cmd YDisplay target set_enabled newval
```

**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()  
display→setLogicalName()display→  
set\_logicalName( )**

**YDisplay**

Changes the logical name of the display.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>nodejs</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YDisplay 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()**  
**display→setOrientation()display→**  
**set\_orientation()**

**YDisplay**

Changes the display orientation.

js	function <b>set_orientation( newval)</b>
nodejs	function <b>set_orientation( newval)</b>
php	function <b>set_orientation( \$newval)</b>
cpp	int <b>set_orientation( Y_ORIENTATION_enum newval)</b>
m	- (int) <b>setOrientation : (Y_ORIENTATION_enum) newval</b>
pas	function <b>set_orientation( newval: Integer): integer</b>
vb	function <b>set_orientation( ByVal newval As Integer) As Integer</b>
cs	int <b>set_orientation( int newval)</b>
java	int <b>set_orientation( int newval)</b>
py	def <b>set_orientation( newval)</b>
cmd	<b>YDisplay 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()**  
**display→setStartupSeq()display→**  
**set\_startupSeq( )**

**YDisplay**

Changes the name of the sequence to play when the displayed is powered on.

<b>js</b>	function <b>set_startupSeq( newval)</b>
<b>node.js</b>	function <b>set_startupSeq( newval)</b>
<b>php</b>	function <b>set_startupSeq( \$newval)</b>
<b>cpp</b>	int <b>set_startupSeq( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setStartupSeq : (NSString*) newval</b>
<b>pas</b>	function <b>set_startupSeq( newval: string): integer</b>
<b>vb</b>	function <b>set_startupSeq( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_startupSeq( string newval)</b>
<b>java</b>	int <b>set_startupSeq( String newval)</b>
<b>py</b>	def <b>set_startupSeq( newval)</b>
<b>cmd</b>	<b>YDisplay 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

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

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

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

**Parameters :**

**data** any kind of object to be stored

**display→stopSequence()display→  
stopSequence( )****YDisplay**

Stops immediately any ongoing sequence replay.

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

The display is left as is.

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**display→swapLayerContent()** **display→**  
**swapLayerContent( )**

YDisplay

Swaps the whole content of two layers.

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	- <b>(int upload : (NSString*) pathname</b> <b>: (NSData*) 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	<b>YDisplay target upload 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');
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 *

### 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, bgcol)

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.

### 3. Reference

#### **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()****YDisplayLayer**

Erases the whole content of the layer (makes it fully transparent).

```
js function clear( )
nodejs function clear( )
php function clear( )
cpp int clear( )
m -(int) clear
pas function clear( ): LongInt
vb function clear( ) As Integer
cs int clear( )
java int clear( )
py def clear( )
cmd YDisplay target [-layer layerId] clear
```

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>
cpp	int <b>clearConsole( )</b>
m	- <b>(int) clearConsole</b>
pas	function <b>clearConsole( ): LongInt</b>
vb	function <b>clearConsole( ) As Integer</b>
cs	int <b>clearConsole( )</b>
java	int <b>clearConsole( )</b>
py	def <b>clearConsole( )</b>
cmd	YDisplay <b>target [-layer layerId] 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( text)</b>
nodejs	function <b>consoleOut( text)</b>
php	function <b>consoleOut( \$text)</b>
cpp	int <b>consoleOut( string text)</b>
m	-(int) <b>consoleOut : (NSString*) text</b>
pas	function <b>consoleOut( text: string): LongInt</b>
vb	function <b>consoleOut( ) As Integer</b>
cs	int <b>consoleOut( string text)</b>
java	int <b>consoleOut( String text)</b>
py	def <b>consoleOut( text)</b>
cmd	<b>YDisplay target [-layer layerId] consoleOut 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()****YDisplayLayer**

Draws a filled rectangular bar at a specified position.

```

js function drawBar( x1, y1, x2, y2)
nodejs function drawBar( x1, y1, x2, y2)
php function drawBar( $x1, $y1, $x2, $y2)
cpp int drawBar( int x1, int y1, int x2, int y2)
m -(int) drawBar : (int) x1
           : (int) y1
           : (int) x2
           : (int) y2
pas function drawBar( x1: LongInt,
                      y1: LongInt,
                      x2: LongInt,
                      y2: LongInt): LongInt
vb function drawBar( ) As Integer
cs int drawBar( int x1, int y1, int x2, int y2)
java int drawBar( int x1, int y1, int x2, int y2)
py def drawBar( x1, y1, x2, y2)
cmd YDisplay target [-layer layerId] drawBar 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→drawBitmap() displaylayer→ drawBitmap( )

YDisplayLayer

Draws a bitmap at the specified position.

```

js   function drawBitmap( x, y, w, bitmap, bgcol)
nodejs function drawBitmap( x, y, w, bitmap, bgcol)
php  function drawBitmap( $x, $y, $w, $bitmap, $bgcol)
cpp   int drawBitmap( int x, int y, int w, string bitmap, int bgcol)
m    -(int) drawBitmap : (int) x
      : (int) y
      : (int) w
      : (NSData*) bitmap
      : (int) bgcol
pas  function drawBitmap( x: LongInt,
                        y: LongInt,
                        w: LongInt,
                        bitmap: TByteArray,
                        bgcol: LongInt): LongInt
vb   procedure drawBitmap( )
cs    int drawBitmap( int x, int y, int w, int bgcol)
java int drawBitmap( int x, int y, int w, int bgcol)
py    def drawBitmap( x, y, w, bitmap, bgcol)
cmd  YDisplay target [-layer layerId] drawBitmap x y w bitmap bgcol

```

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 drawCircle( x, y, r)
node.js function drawCircle( x, y, r)
php function drawCircle( $x, $y, $r)
cpp int drawCircle( int x, int y, int r)
m -(int) drawCircle : (int) x
               : (int) y
               : (int) r
pas function drawCircle( x: LongInt, y: LongInt, r: LongInt): LongInt
vb function drawCircle( ) As Integer
cs int drawCircle( int x, int y, int r)
java int drawCircle( int x, int y, int r)
py def drawCircle( x, y, r)
cmd YDisplay target [-layer layerId] drawCircle x y r
```

#### 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.

**displaylayer→drawDisc()displaylayer→  
drawDisc( )**
**YDisplayLayer**

Draws a filled disc at a given position.

<b>js</b>	function <b>drawDisc( x, y, r)</b>
<b>nodejs</b>	function <b>drawDisc( x, y, r)</b>
<b>php</b>	function <b>drawDisc( \$x, \$y, \$r)</b>
<b>cpp</b>	int <b>drawDisc( int x, int y, int r)</b>
<b>m</b>	- <b>(int) drawDisc : (int) x</b> <b>: (int) y</b> <b>: (int) r</b>
<b>pas</b>	function <b>drawDisc( x: LongInt, y: LongInt, r: LongInt): LongInt</b>
<b>vb</b>	function <b>drawDisc( ) As Integer</b>
<b>cs</b>	int <b>drawDisc( int x, int y, int r)</b>
<b>java</b>	int <b>drawDisc( int x, int y, int r)</b>
<b>py</b>	def <b>drawDisc( x, y, r)</b>
<b>cmd</b>	<b>YDisplay target [-layer layerId] drawDisc x y r</b>

**Parameters :**

- x** the distance from left of layer to the center of the disc, in pixels
- y** the distance from top of layer to the center of the disc, in pixels
- r** the radius of the disc, in pixels

**Returns :**

**YAPI\_SUCCESS** if the call succeeds.

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 drawImage( x, y, imagename)
node.js function drawImage( x, y, imagename)
php  function drawImage( $x, $y, $imagename)
cpp   int drawImage( int x, int y, string imagename)
m     -(int) drawImage : (int) x
          : (int) y
          : (NSString*) imagename
pas   function drawImage( x: LongInt, y: LongInt, imagename: string): LongInt
vb    function drawImage( ) As Integer
cs    int drawImage( int x, int y, string imagename)
java  int drawImage( int x, int y, String imagename)
py    def drawImage( x, y, imagename)
cmd   YDisplay target [-layer layerId] drawImage x y imagename

```

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 :

**x** the distance from left of layer to the left of the image, in pixels  
**y** the distance from top of layer to the top of the image, in pixels  
**imagename** the GIF file name

#### Returns :

`YAPI_SUCCESS` if the call succeeds.

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

**displaylayer→drawPixel()  
displaylayer→  
drawPixel()****YDisplayLayer**

Draws a single pixel at the specified position.

```
js function drawPixel( x, y)
nodejs function drawPixel( x, y)
php function drawPixel( $x, $y)
cpp int drawPixel( int x, int y)
m -(int) drawPixel : (int) x
                : (int) y
pas function drawPixel( x: LongInt, y: LongInt): LongInt
vb function drawPixel( ) As Integer
cs int drawPixel( int x, int y)
java int drawPixel( int x, int y)
py def drawPixel( x, y)
cmd YDisplay target [-layer layerId] drawPixel x y
```

**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 <b>drawRect( x1, y1, x2, y2)</b>
node.js	function <b>drawRect( x1, y1, x2, y2)</b>
php	function <b>drawRect( \$x1, \$y1, \$x2, \$y2)</b>
cpp	<b>int drawRect( int x1, int y1, int x2, int y2)</b>
m	- <b>(int) drawRect : (int) x1                   : (int) y1                   : (int) x2                   : (int) y2</b>
pas	function <b>drawRect( x1: LongInt,                   y1: LongInt,                   x2: LongInt,                   y2: LongInt): LongInt</b>
vb	function <b>drawRect( ) As Integer</b>
cs	<b>int drawRect( int x1, int y1, int x2, int y2)</b>
java	<b>int drawRect( int x1, int y1, int x2, int y2)</b>
py	<b>def drawRect( x1, y1, x2, y2)</b>
cmd	<b>YDisplay target [-layer layerId] drawRect x1 y1 x2 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→drawText()displaylayer→  
drawText( )****YDisplayLayer**

Draws a text string at the specified position.

```

js   function drawText( x, y, anchor, text)
nodejs function drawText( x, y, anchor, text)
php  function drawText( $x, $y, $anchor, $text)
cpp   int drawText( int x, int y, Y_ALIGN anchor, string text)
m    -(int) drawText : (int) x
      : (int) y
      : (Y_ALIGN) anchor
      : (NSString*) text

pas  function drawText( x: LongInt,
                      y: LongInt,
                      anchor: TYALIGN,
                      text: string): LongInt

vb   function drawText( ) As Integer
cs    int drawText( int x, int y, ALIGN anchor, string text)
java   int drawText( int x, int y, ALIGN anchor, String text)
py    def drawText( x, y, anchor, text)
cmd   YDisplay target [-layer layerId] drawText x y anchor text

```

The point of the text that is aligned to the specified pixel position is called the anchor point, and can be chosen among several options. Text is rendered from left to right, without implicit wrapping.

**Parameters :**

**x** the distance from left of layer to the text anchor point, in pixels  
**y** the distance from top of layer to the text anchor point, in pixels  
**anchor** the text anchor point, chosen among the Y\_ALIGN enumeration: Y\_ALIGN\_TOP\_LEFT,  
Y\_ALIGN\_CENTER\_LEFT, Y\_ALIGN\_BASELINE\_LEFT, Y\_ALIGN\_BOTTOM\_LEFT,  
Y\_ALIGN\_TOP\_CENTER, Y\_ALIGN\_CENTER, Y\_ALIGN\_BASELINE\_CENTER,  
Y\_ALIGN\_BOTTOM\_CENTER, Y\_ALIGN\_TOP\_DECIMAL,  
Y\_ALIGN\_CENTER\_DECIMAL, Y\_ALIGN\_BASELINE\_DECIMAL,  
Y\_ALIGN\_BOTTOM\_DECIMAL, Y\_ALIGN\_TOP\_RIGHT, Y\_ALIGN\_CENTER\_RIGHT,  
Y\_ALIGN\_BASELINE\_RIGHT, Y\_ALIGN\_BOTTOM\_RIGHT.  
**text** the text string to draw

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**displaylayer→get\_display()**  
**displaylayer→display()displaylayer→**  
**get\_display( )**

**YDisplayLayer**

Gets parent YDisplay.

```
js function get_display( )
nodejs function get_display( )
php function get_display( )
cpp YDisplay* get_display( )
m -(YDisplay*) display
pas function get_display( ): TYDisplay
vb function get_display( ) As YDisplay
cs YDisplay get_display( )
java YDisplay get_display( )
py def get_display( )
```

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.

<b>js</b>	function <b>get_displayHeight( )</b>
<b>nodejs</b>	function <b>get_displayHeight( )</b>
<b>php</b>	function <b>get_displayHeight( )</b>
<b>cpp</b>	int <b>get_displayHeight( )</b>
<b>m</b>	-(int) <b>displayHeight</b>
<b>pas</b>	function <b>get_displayHeight( ): LongInt</b>
<b>vb</b>	function <b>get_displayHeight( ) As Integer</b>
<b>cs</b>	int <b>get_displayHeight( )</b>
<b>java</b>	int <b>get_displayHeight( )</b>
<b>py</b>	def <b>get_displayHeight( )</b>
<b>cmd</b>	<b>YDisplay target [-layer layerId] 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()**  
**displaylayer→displayWidth()displaylayer→**  
**get\_displayWidth( )**

**YDisplayLayer**

Returns the display width, in pixels.

<b>js</b>	function <b>get_displayWidth( )</b>
<b>nodejs</b>	function <b>get_displayWidth( )</b>
<b>php</b>	function <b>get_displayWidth( )</b>
<b>cpp</b>	int <b>get_displayWidth( )</b>
<b>m</b>	-(int) <b>displayWidth</b>
<b>pas</b>	function <b>get_displayWidth( ): LongInt</b>
<b>vb</b>	function <b>get_displayWidth( ) As Integer</b>
<b>cs</b>	int <b>get_displayWidth( )</b>
<b>java</b>	int <b>get_displayWidth( )</b>
<b>py</b>	def <b>get_displayWidth( )</b>
<b>cmd</b>	YDisplay <b>target [-layer layerId] 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()**  
**displaylayer→layerHeight()** **displaylayer→get\_layerHeight( )**

**YDisplayLayer**

Returns the height of the layers to draw on, in pixels.

js	function <b>get_layerHeight( )</b>
node.js	function <b>get_layerHeight( )</b>
php	function <b>get_layerHeight( )</b>
cpp	int <b>get_layerHeight( )</b>
m	-(int) <b>layerHeight</b>
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	<b>YDisplay target [-layer layerId] 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()**  
**displaylayer→layerWidth()displaylayer→**  
**get\_layerWidth( )**

**YDisplayLayer**

Returns the width of the layers to draw on, in pixels.

```
js function get_layerWidth( )  
nodejs function get_layerWidth( )  
php function get_layerWidth( )  
cpp int get_layerWidth( )  
m -(int) layerWidth  
pas function get_layerWidth( ): LongInt  
vb function get_layerWidth( ) As Integer  
cs int get_layerWidth( )  
java int get_layerWidth( )  
py def get_layerWidth( )  
cmd YDisplay target [-layer layerId] get_layerWidth
```

**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()****YDisplayLayer**

Hides the layer.

```
js function hide( )
node.js function hide( )
php function hide( )
cpp int hide( )
m -(int) hide
pas function hide( ): LongInt
vb function hide( ) As Integer
cs int hide( )
java int hide( )
py def hide( )
cmd YDisplay target [-layer layerId] hide
```

The state of the layer is preserved 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 lineTo( x, y)
nodejs function lineTo( x, y)
php function lineTo( $x, $y)
cpp int lineTo( int x, int y)
m -(int) lineTo : (int) x
: (int) y
pas function lineTo( x: LongInt, y: LongInt): LongInt
vb function lineTo( ) As Integer
cs int lineTo( int x, int y)
java int lineTo( int x, int y)
py def lineTo( x, y)
cmd YDisplay target [-layer layerId] lineTo x y
```

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.

**displaylayer→moveTo()displaylayer→moveTo( )****YDisplayLayer**

Moves the drawing pointer of this layer to the specified position.

js	function <b>moveTo( x, y)</b>
nodejs	function <b>moveTo( x, y)</b>
php	function <b>moveTo( \$x, \$y)</b>
cpp	int <b>moveTo( int x, int y)</b>
m	- <b>(int) moveTo : (int) x</b> <b>: (int) y</b>
pas	function <b>moveTo( x: LongInt, y: LongInt): LongInt</b>
vb	function <b>moveTo( ) As Integer</b>
cs	int <b>moveTo( int x, int y)</b>
java	int <b>moveTo( int x, int y)</b>
py	def <b>moveTo( x, y)</b>
cmd	<b>YDisplay target [-layer layerId] moveTo 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 reset( )  
nodejs function reset( )  
php function reset( )  
cpp int reset( )  
m -(int) reset  
pas function reset( ): LongInt  
vb function reset( ) As Integer  
cs int reset( )  
java int reset( )  
py def reset( )  
cmd YDisplay target [-layer layerId] reset
```

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 selectColorPen( color)
nodejs function selectColorPen( color)
php function selectColorPen( $color)
cpp int selectColorPen( int color)
m -(int) selectColorPen : (int) color
pas function selectColorPen( color: LongInt): LongInt
vb function selectColorPen( ) As Integer
cs int selectColorPen( int color)
java int selectColorPen( int color)
py def selectColorPen( color)
cmd YDisplay target [-layer layerId] selectColorPen color
```

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 selectEraser( )
nodejs function selectEraser( )
php function selectEraser( )
cpp int selectEraser( )
m -(int) selectEraser
pas function selectEraser( ): LongInt
vb function selectEraser( ) As Integer
cs int selectEraser( )
java int selectEraser( )
py def selectEraser( )
cmd YDisplay target [-layer layerId] selectEraser
```

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( fontname)</b>
nodejs	function <b>selectFont( fontname)</b>
php	function <b>selectFont( \$fontname)</b>
cpp	int <b>selectFont( string fontname)</b>
m	- <b>(int) selectFont : (NSString*) fontname</b>
pas	function <b>selectFont( fontname: string): LongInt</b>
vb	function <b>selectFont( ) As Integer</b>
cs	int <b>selectFont( string fontname)</b>
java	int <b>selectFont( String fontname)</b>
py	def <b>selectFont( fontname)</b>
cmd	<b>YDisplay target [-layer layerId] selectFont 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 selectGrayPen( graylevel)
node.js function selectGrayPen( graylevel)
php function selectGrayPen( $graylevel)
cpp int selectGrayPen( int graylevel)
m -(int) selectGrayPen : (int) graylevel
pas function selectGrayPen( graylevel: LongInt): LongInt
vb function selectGrayPen( ) As Integer
cs int selectGrayPen( int graylevel)
java int selectGrayPen( int graylevel)
py def selectGrayPen( graylevel)
cmd YDisplay target [-layer layerId] selectGrayPen graylevel
```

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> ( mode)
node.js	function <b>setAntialiasingMode</b> ( mode)
php	function <b>setAntialiasingMode</b> ( \$mode)
cpp	int <b>setAntialiasingMode</b> ( bool mode)
m	-(int) <b>setAntialiasingMode</b> : (bool) mode
pas	function <b>setAntialiasingMode</b> ( mode: boolean): LongInt
vb	function <b>setAntialiasingMode</b> ( ) As Integer
cs	int <b>setAntialiasingMode</b> ( bool mode)
java	int <b>setAntialiasingMode</b> ( boolean mode)
py	def <b>setAntialiasingMode</b> ( mode)
cmd	YDisplay target [-layer layerId] <b>setAntialiasingMode</b> mode

Anti-aliasing provides a smoother aspect when looked from far enough, but it can add fuzziness 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()****YDisplayLayer****displaylayer→setConsoleBackground( )**

Sets up the background color used by the `clearConsole` function and by the console scrolling feature.

```
js   function setConsoleBackground( bgcol)
nodejs function setConsoleBackground( bgcol)
php  function setConsoleBackground( $bgcol)
cpp   int setConsoleBackground( int bgcol)
m    -(int) setConsoleBackground : (int) bgcol
pas   function setConsoleBackground( bgcol: LongInt): LongInt
vb    function setConsoleBackground( ) As Integer
cs    int setConsoleBackground( int bgcol)
java  int setConsoleBackground( int bgcol)
py    def setConsoleBackground( bgcol)
cmd   YDisplay target [-layer layerId] setConsoleBackground bgcol
```

**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.

<code>js</code>	<code>function setConsoleMargins( x1, y1, x2, y2)</code>
<code>nodejs</code>	<code>function setConsoleMargins( x1, y1, x2, y2)</code>
<code>php</code>	<code>function setConsoleMargins( \$x1, \$y1, \$x2, \$y2)</code>
<code>cpp</code>	<code>int setConsoleMargins( int x1, int y1, int x2, int y2)</code>
<code>m</code>	<code>- (int) setConsoleMargins : (int) x1                           : (int) y1                           : (int) x2                           : (int) y2</code>
<code>pas</code>	<code>function setConsoleMargins( x1: LongInt,                               y1: LongInt,                               x2: LongInt,                               y2: LongInt): LongInt</code>
<code>vb</code>	<code>function setConsoleMargins( ) As Integer</code>
<code>cs</code>	<code>int setConsoleMargins( int x1, int y1, int x2, int y2)</code>
<code>java</code>	<code>int setConsoleMargins( int x1, int y1, int x2, int y2)</code>
<code>py</code>	<code>def setConsoleMargins( x1, y1, x2, y2)</code>
<code>cmd</code>	<code>YDisplay target [-layer layerId] setConsoleMargins x1 y1 x2 y2</code>

### 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( wordwrap)</b>
node.js	function <b>setConsoleWordWrap( wordwrap)</b>
php	function <b>setConsoleWordWrap( \$wordwrap)</b>
cpp	int <b>setConsoleWordWrap( bool wordwrap)</b>
m	- <b>(int) setConsoleWordWrap : (bool) wordwrap</b>
pas	function <b>setConsoleWordWrap( wordwrap: boolean): LongInt</b>
vb	function <b>setConsoleWordWrap( ) As Integer</b>
cs	int <b>setConsoleWordWrap( bool wordwrap)</b>
java	int <b>setConsoleWordWrap( boolean wordwrap)</b>
py	def <b>setConsoleWordWrap( wordwrap)</b>
cmd	<b>YDisplay target [-layer layerId] setConsoleWordWrap 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 setLayerPosition( x, y, scrollTime)
nodejs function setLayerPosition( x, y, scrollTime)
php  function setLayerPosition( $x, $y, $scrollTime)
cpp   int setLayerPosition( int x, int y, int scrollTime)
m     -(int) setLayerPosition : (int) x
          : (int) y
          : (int) scrollTime
pas   function setLayerPosition( x: LongInt,
                                y: LongInt,
                                scrollTime: LongInt): LongInt
vb    function setLayerPosition( ) As Integer
cs    int setLayerPosition( int x, int y, int scrollTime)
java  int setLayerPosition( int x, int y, int scrollTime)
py    def setLayerPosition( x, y, scrollTime)
cmd   YDisplay target [-layer layerId] setLayerPosition x y scrollTime

```

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 :

- x** the distance from left of display to the upper left corner of the layer
- y** the distance from top of display to the upper left corner of the layer
- scrollTime** 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 unhide( )
nodejs function unhide( )
php function unhide( )
cpp int unhide( )
m -(int) unhide
pas function unhide( ): LongInt
vb function unhide( ) As Integer
cs int unhide( )
java int unhide( )
py def unhide( )
cmd YDisplay target [-layer layerId] unhide
```

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>
node.js	var yoctolib = require('yoctolib');
php	var YDualPower = yoctolib.YDualPower;
require_once('yocto_dualpower.php');	
cpp	#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()

### 3. Reference

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() yFindDualPower()yFindDualPower( )

**YDualPower**

Retrieves a dual power control for a given identifier.

js	function <b>yFindDualPower( func)</b>
node.js	function <b>FindDualPower( func)</b>
php	function <b>yFindDualPower( \$func)</b>
cpp	YDualPower* <b>yFindDualPower( const string&amp; func)</b>
m	YDualPower* <b>yFindDualPower( NSString* func)</b>
pas	function <b>yFindDualPower( func: string): TYDualPower</b>
vb	function <b>yFindDualPower( ByVal func As String) As YDualPower</b>
cs	YDualPower <b>FindDualPower( string func)</b>
java	YDualPower <b>FindDualPower( String func)</b>
py	def <b>FindDualPower( 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() yFirstDualPower()yFirstDualPower( )

YDualPower

Starts the enumeration of dual power controls currently accessible.

js	function <b>yFirstDualPower()</b>
node.js	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()****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)=RELAYL01-123456.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 power control (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**dualpower→get\_advertisedValue()**  
**dualpower→advertisedValue()****dualpower→get\_advertisedValue( )**

**YDualPower**

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

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

**YDualPower**

Returns the error message of the latest error with the power control.

<b>js</b>	function <b>getErrorMessage( )</b>
<b>nodejs</b>	function <b>getErrorMessage( )</b>
<b>php</b>	function <b>getErrorMessage( )</b>
<b>cpp</b>	string <b>getErrorMessage( )</b>
<b>m</b>	-(NSString*) <b>errorMessage</b>
<b>pas</b>	function <b>getErrorMessage( )</b> : string
<b>vb</b>	function <b>getErrorMessage( )</b> As String
<b>cs</b>	string <b>getErrorMessage( )</b>
<b>java</b>	String <b>getErrorMessage( )</b>
<b>py</b>	def <b>getErrorMessage( )</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()**  
**dualpower→errorType()** dualpower→  
**get\_errorType( )**

**YDualPower**

Returns the numerical error code of the latest error with the power control.

**js** function **get\_errorType( )**  
**nodejs** function **get\_errorType( )**  
**php** function **get\_errorType( )**  
**cpp** YRETCODE **get\_errorType( )**  
**pas** function **get\_errorType( )**: YRETCODE  
**vb** function **get\_errorType( )** As YRETCODE  
**cs** YRETCODE **get\_errorType( )**  
**java** int **get\_errorType( )**  
**py** def **get\_errorType( )**

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the power control object

**dualpower→get\_extVoltage()**  
**dualpower→extVoltage()****dualpower→get\_extVoltage( )**

**YDualPower**

Returns the measured voltage on the external power source, in millivolts.

<b>js</b>	function <b>get_extVoltage( )</b>
<b>nodejs</b>	function <b>get_extVoltage( )</b>
<b>php</b>	function <b>get_extVoltage( )</b>
<b>cpp</b>	int <b>get_extVoltage( )</b>
<b>m</b>	-(int) <b>extVoltage</b>
<b>pas</b>	function <b>get_extVoltage( )</b> : LongInt
<b>vb</b>	function <b>get_extVoltage( )</b> As Integer
<b>cs</b>	int <b>get_extVoltage( )</b>
<b>java</b>	int <b>get_extVoltage( )</b>
<b>py</b>	def <b>get_extVoltage( )</b>
<b>cmd</b>	<b>YDualPower target 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()**  
**dualpower→friendlyName()****dualpower→get\_friendlyName( )**

**YDualPower**

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

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

The returned string uses the logical names of the module and of the power control if they are defined, otherwise the serial number of the module and the hardware identifier of the power control (for exemple: 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>
node.js	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.

**dualpower→get\_functionId()**  
**dualpower→functionId()****dualpower→get\_functionId( )**

**YDualPower**

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()**  
**dualpower→hardwareId()****dualpower→get\_hardwareId( )**

**YDualPower**

Returns the unique hardware identifier of the power control in the form SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the power control (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**dualpower→get\_logicalName()**  
**dualpower→logicalName()****dualpower→get\_logicalName( )**

**YDualPower**

Returns the logical name of the power control.

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

**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 online.

**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)`  
`node.js` `function get_module_async( callback, context)`

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**dualpower→get\_powerControl()**  
**dualpower→powerControl()****dualpower→**  
**get\_powerControl()**

**YDualPower**

Returns the selected power source for module functions that require lots of current.

<b>js</b>	function <b>get_powerControl( )</b>
<b>nodejs</b>	function <b>get_powerControl( )</b>
<b>php</b>	function <b>get_powerControl( )</b>
<b>cpp</b>	Y_POWERCONTROL_enum <b>get_powerControl( )</b>
<b>m</b>	-(Y_POWERCONTROL_enum) powerControl
<b>pas</b>	function <b>get_powerControl( ): Integer</b>
<b>vb</b>	function <b>get_powerControl( ) As Integer</b>
<b>cs</b>	int <b>get_powerControl( )</b>
<b>java</b>	int <b>get_powerControl( )</b>
<b>py</b>	def <b>get_powerControl( )</b>
<b>cmd</b>	<b>YDualPower target 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()**  
**dualpower→powerState()****dualpower→**  
**get\_powerState( )**

**YDualPower**

Returns the current power source for module functions that require lots of current.

js	function get_powerState( )
nodejs	function get_powerState( )
php	function get_powerState( )
cpp	Y_POWERSTATE_enum get_powerState( )
m	-(Y_POWERSTATE_enum) powerState
pas	function get_powerState( ): Integer
vb	function get_powerState( ) As Integer
cs	int get_powerState( )
java	int get_powerState( )
py	def get_powerState( )
cmd	YDualPower target get_powerState

**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)**  
**dualpower**→**userData()****dualpower**→  
**get(userData)**

**YDualPower**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	function <b>get(userData)</b>
node.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.

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

<code>js</code>	<code>function nextDualPower( )</code>
<code>node.js</code>	<code>function nextDualPower( )</code>
<code>php</code>	<code>function nextDualPower( )</code>
<code>cpp</code>	<code>YDualPower * nextDualPower( )</code>
<code>m</code>	<code>-(YDualPower*) nextDualPower</code>
<code>pas</code>	<code>function nextDualPower( ): TYDualPower</code>
<code>vb</code>	<code>function nextDualPower( ) As YDualPower</code>
<code>cs</code>	<code>YDualPower nextDualPower( )</code>
<code>java</code>	<code>YDualPower nextDualPower( )</code>
<code>py</code>	<code>def nextDualPower( )</code>

**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.

<code>js</code>	<code>function registerValueCallback( callback)</code>
<code>node.js</code>	<code>function registerValueCallback( callback)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>int registerValueCallback( YDualPowerValueCallback callback)</code>
<code>m</code>	<code>-(int) registerValueCallback : (YDualPowerValueCallback) callback</code>
<code>pas</code>	<code>function registerValueCallback( callback: TYDualPowerValueCallback): LongInt</code>
<code>vb</code>	<code>function registerValueCallback( ) As Integer</code>
<code>cs</code>	<code>int registerValueCallback( ValueCallback callback)</code>
<code>java</code>	<code>int registerValueCallback( UpdateCallback callback)</code>
<code>py</code>	<code>def registerValueCallback( callback)</code>

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()**  
**dualpower→setLogicalName()****dualpower→set\_logicalName( )**

**YDualPower**

Changes the logical name of the power control.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the power control.

**Returns :**

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

**dualpower→set\_powerControl()**  
**dualpower→setPowerControl()**  
**dualpower→set\_powerControl()**

**YDualPower**

Changes the selected power source for module functions that require lots of current.

<b>js</b>	function <b>set_powerControl( newval)</b>
<b>nodejs</b>	function <b>set_powerControl( newval)</b>
<b>php</b>	function <b>set_powerControl( \$newval)</b>
<b>cpp</b>	int <b>set_powerControl( Y_POWERCONTROL_enum newval)</b>
<b>m</b>	-(int) <b>setPowerControl : (Y_POWERCONTROL_enum) newval</b>
<b>pas</b>	function <b>set_powerControl( newval: Integer): integer</b>
<b>vb</b>	function <b>set_powerControl( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_powerControl( int newval)</b>
<b>java</b>	int <b>set_powerControl( int newval)</b>
<b>py</b>	def <b>set_powerControl( newval)</b>
<b>cmd</b>	<b>YDualPower 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)**

**YDualPower**

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

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData)</b>
vb	procedure <b>set(userData)</b> ByVal <b>data As Object</b>
cs	void <b>set(userData)</b>
java	void <b>set(userData)</b>
py	def <b>set(userData)</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');
cpp	#include "yocto_files.h"
m	#import "yocto_files.h"
pas	uses yocto_files;
vb	yocto_files.vb
cs	yocto_files.cs
java	import com.yoctopuce.YoctoAPI.YFiles;
py	from yocto_files import *

### Global functions

#### yFindFiles(func)

Retrieves a filesystem for a given identifier.

#### yFirstFiles()

Starts the enumeration of filesystems currently accessible.

### YFiles methods

#### files→describe()

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

#### files→download(pathname)

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

#### files→download\_async(pathname, callback, context)

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

#### files→format\_fs()

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

#### files→get\_advertisedValue()

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

#### files→get\_errorMessage()

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

#### files→get\_errorType()

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

#### files→get\_filesCount()

Returns the number of files currently loaded in the filesystem.

#### files→get\_freeSpace()

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

#### files→get\_friendlyName()

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

#### files→get\_functionDescriptor()

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

#### files→get\_functionId()

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

**files→get\_hardwareId()**

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

**files→get\_list(pattern)**

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

**files→get\_logicalName()**

Returns the logical name of the filesystem.

**files→get\_module()**

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

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

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

**files→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set(userData).

**files→isOnline()**

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

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

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

**files→load(msValidity)**

Preloads the filesystem cache with a specified validity duration.

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

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

**files→nextFiles()**

Continues the enumeration of filesystems started using yFirstFiles( ).

**files→registerValueCallback(callback)**

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

**files→remove(pathname)**

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

**files→set\_logicalName(newval)**

Changes the logical name of the filesystem.

**files→set\_userData(data)**

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

**files→upload(pathname, content)**

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

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

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

**YFiles.FindFiles()****YFiles****yFindFiles()yFindFiles( )**

Retrieves a filesystem for a given identifier.

<b>js</b>	<code>function yFindFiles( func)</code>
<b>node.js</b>	<code>function FindFiles( func)</code>
<b>php</b>	<code>function yFindFiles( \$func)</code>
<b>cpp</b>	<code>YFiles* yFindFiles( string func)</code>
<b>m</b>	<code>+ (YFiles*) yFindFiles : (NSString*) func</code>
<b>pas</b>	<code>function yFindFiles( func: string): TYFiles</code>
<b>vb</b>	<code>function yFindFiles( ByVal func As String) As YFiles</code>
<b>cs</b>	<code>YFiles FindFiles( string func)</code>
<b>java</b>	<code>YFiles FindFiles( String func)</code>
<b>py</b>	<code>def FindFiles( 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 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()yFirstFiles( )**

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()****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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**files→download()files→download()****YFiles**

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

js	function <b>download( pathname)</b>
node.js	function <b>download( pathname)</b>
php	function <b>download( \$pathname)</b>
cpp	string <b>download( string pathname)</b>
m	- <b>(NSData*) download : (NSString*) pathname</b>
pas	function <b>download( pathname: string): TByteArray</b>
vb	function <b>download( ) As Byte</b>
py	<b>def download( pathname)</b>
cmd	<b>YFiles 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 download is completed. 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	- <b>(int) 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( )**

**YFiles**

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

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

**Returns :**

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

**files→getErrorMessage()**

YFiles

**files→errorMessage()files→getErrorMessage( )**

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

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

**YFiles**  
**files→get\_errorType()**  
**files→errorType()files→get\_errorType( )**

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

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

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

**Returns :**

a number corresponding to the code of the latest error that 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( ): LongInt</b>
vb	function <b>get_filesCount( ) As Integer</b>
cs	int <b>get_filesCount( )</b>
java	int <b>get_filesCount( )</b>
py	def <b>get_filesCount( )</b>
cmd	<b>YFiles target 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()** YFiles  
**files→freeSpace()files→get\_freeSpace( )**

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

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

**Returns :**

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

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

**files→get\_friendlyName()****YFiles****files→friendlyName()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 exemple: MyCustomName . relay1)

**Returns :**

a string that uniquely identifies the filesystem using logical names (ex: MyCustomName . relay1) On failure, throws an exception or returns Y\_FRIENDLYNAME\_INVALID.

**files→get\_functionDescriptor()**  
**files→functionDescriptor()files→**  
**get\_functionDescriptor( )**

**YFiles**

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

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	String <b>get_functionDescriptor( )</b>
<b>py</b>	<b>def 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( ) As String</b>
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()** YFiles  
**files→hardwareId()files→get\_hardwareId( )**

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

js	function <b>get_hardwareId( )</b>
node.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( ) As String</b>
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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the filesystem (ex: RELAYL01-123456.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.

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

Returns the logical name of the filesystem.

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

**Returns :**

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

**files→get\_module()****YFiles****files→module()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	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 online.

**Returns :**

an instance of YModule

**files→get\_module\_async()**  
**files→module\_async()****YFiles**

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

`js` `function get_module_async( callback, context)`  
`node.js` `function get_module_async( callback, context)`

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**files→get(userData)****YFiles****files→userData()files→get(userData())**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	<code>function get(userData) { ... }</code>
nodejs	<code>function get(userData) { ... }</code>
php	<code>function get(userData) { ... }</code>
cpp	<code>void * get(userData) { ... }</code>
m	<code>-(void*)(userData)</code>
pas	<code>function get(userData): Tobject { ... }</code>
vb	<code>function get(userData) As Object { ... }</code>
cs	<code>object get(userData) { ... }</code>
java	<code>Object get(userData) { ... }</code>
py	<code>def get(userData) { ... }</code>

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

**Returns :**

the object stored previously by the caller.

**files→isOnline()****YFiles**

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

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) 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()****YFiles**

Preloads the filesystem cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-( <b>YRETCODE</b> ) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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 registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php  function registerValueCallback( $callback)
cpp   int registerValueCallback( YFilesValueCallback callback)
m    -(int) registerValueCallback : (YFilesValueCallback) callback
pas   function registerValueCallback( callback: TYFilesValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs   int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
py    def registerValueCallback( callback)
```

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

**Parameters :**

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

**files→remove()****YFiles**

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

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

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

**Parameters :**

**pathname** path and name of the file to remove.

**Returns :**

`YAPI_SUCCESS` if the call succeeds.

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

**files→set\_logicalName()**  
**files→setLogicalName()****files→**  
**set\_logicalName( )**

**YFiles**

Changes the logical name of the filesystem.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YFiles 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)****YFiles****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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData)</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> )
node.js	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	- <b>(int upload : (NSString*) pathname</b> <b>: (NSData*) 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 target <b>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_geneticsensor.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YGenericSensor = yoctolib.YGenericSensor;
php	require_once('yocto_geneticsensor.php');
cpp	#include "yocto_geneticsensor.h"
m	#import "yocto_geneticsensor.h"
pas	uses yocto_geneticsensor;
vb	yocto_geneticsensor.vb
cs	yocto_geneticsensor.cs
java	import com.yoctopuce.YoctoAPI.YGenericSensor;
py	from yocto_geneticsensor import *

### Global functions

#### yFindGenericSensor(func)

Retrieves a generic sensor for a given identifier.

#### yFirstGenericSensor()

Starts the enumeration of generic sensors currently accessible.

### YGenericSensor methods

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

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

#### geneticsensor→describe()

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

#### geneticsensor→get\_advertisedValue()

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

#### geneticsensor→get\_currentRawValue()

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

#### geneticsensor→get\_currentValue()

Returns the current measured value.

#### geneticsensor→get\_errorMessage()

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

#### geneticsensor→get\_errorType()

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

#### geneticsensor→get\_friendlyName()

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

#### geneticsensor→get\_functionDescriptor()

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

#### geneticsensor→get\_functionId()

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

#### geneticsensor→get\_hardwareId()

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

<b>genericsensor→get_highestValue()</b>	Returns the maximal value observed for the measure since the device was started.
<b>genericsensor→get_logFrequency()</b>	Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>genericsensor→get_logicalName()</b>	Returns the logical name of the generic sensor.
<b>genericsensor→get_lowestValue()</b>	Returns the minimal value observed for the measure since the device was started.
<b>genericsensor→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>genericsensor→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>genericsensor→get_recordedData(startTime, endTime)</b>	Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>genericsensor→get_reportFrequency()</b>	Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>genericsensor→get_resolution()</b>	Returns the resolution of the measured values.
<b>genericsensor→get_signalRange()</b>	Returns the electric signal range used by the sensor.
<b>genericsensor→get_signalUnit()</b>	Returns the measuring unit of the electrical signal used by the sensor.
<b>genericsensor→get_signalValue()</b>	Returns the measured value of the electrical signal used by the sensor.
<b>genericsensor→get_unit()</b>	Returns the measuring unit for the measure.
<b>genericsensor→get(userData)</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>genericsensor→get_valueRange()</b>	Returns the physical value range measured by the sensor.
<b>genericsensor→isOnline()</b>	Checks if the generic sensor is currently reachable, without raising any error.
<b>genericsensor→isOnline_async(callback, context)</b>	Checks if the generic sensor is currently reachable, without raising any error (asynchronous version).
<b>genericsensor→load(msValidity)</b>	Preloads the generic sensor cache with a specified validity duration.
<b>genericsensor→loadCalibrationPoints(rawValues, refValues)</b>	Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>genericsensor→load_async(msValidity, callback, context)</b>	Preloads the generic sensor cache with a specified validity duration (asynchronous version).
<b>genericsensor→nextGenericSensor()</b>	Continues the enumeration of generic sensors started using yFirstGenericSensor( ).
<b>genericsensor→registerTimedReportCallback(callback)</b>	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()yFindGenericSensor( )

**YGenericSensor**

Retrieves a generic sensor for a given identifier.

```
js function yFindGenericSensor( func)
node.js function FindGenericSensor( func)
php function yFindGenericSensor( $func)
cpp YGenericSensor* yFindGenericSensor( const string& func)
m YGenericSensor* yFindGenericSensor( NSString* func)
pas function yFindGenericSensor( func: string): TYGenericSensor
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()yFirstGenericSensor( )****YGenericSensor**

Starts the enumeration of generic sensors currently accessible.

```
js function yFirstGenericSensor( )
nodejs function FirstGenericSensor( )
php function yFirstGenericSensor( )
cpp YGenericSensor* yFirstGenericSensor( )
m YGenericSensor* yFirstGenericSensor( )
pas function yFirstGenericSensor( ): TYGenericSensor
vb function yFirstGenericSensor( ) As YGenericSensor
cs YGenericSensor FirstGenericSensor( )
java YGenericSensor FirstGenericSensor( )
py def FirstGenericSensor( )
```

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()****YGenericSensor****genericsensor→calibrateFromPoints( )**

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	- <b>(NSString*) 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)=RELAYL01-123456.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 generic sensor (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.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 **get\_advertisedValue( )**  
**nodejs** function **get\_advertisedValue( )**  
**php** function **get\_advertisedValue( )**  
**cpp** string **get\_advertisedValue( )**  
**m** -(NSString\*) advertisedValue  
**pas** function **get\_advertisedValue( )**: string  
**vb** function **get\_advertisedValue( )** As String  
**cs** string **get\_advertisedValue( )**  
**java** String **get\_advertisedValue( )**  
**py** def **get\_advertisedValue( )**  
**cmd** YGenericSensor **target get\_advertisedValue**

**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()**  
**genericsensor→currentRawValue()genericsensor**  
**→get\_currentRawValue( )**

**YGenericSensor**

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

<b>js</b>	function <b>get_currentRawValue( )</b>
<b>nodejs</b>	function <b>get_currentRawValue( )</b>
<b>php</b>	function <b>get_currentRawValue( )</b>
<b>cpp</b>	double <b>get_currentRawValue( )</b>
<b>m</b>	-(double) currentRawValue
<b>pas</b>	function <b>get_currentRawValue( ): double</b>
<b>vb</b>	function <b>get_currentRawValue( ) As Double</b>
<b>cs</b>	double <b>get_currentRawValue( )</b>
<b>java</b>	double <b>get_currentRawValue( )</b>
<b>py</b>	def <b>get_currentRawValue( )</b>
<b>cmd</b>	<b>YGenericSensor target 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 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→getErrorMessage()**  
**genericsensor→errorMessage()** genericsensor→  
**getErrorMessage( )**

**YGenericSensor**

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

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

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()**  
**genericsensor→errorType()****genericsensor→get\_errorType( )**

**YGenericSensor**

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

**js** function **get\_errorType( )**  
**nodejs** function **get\_errorType( )**  
**php** function **get\_errorType( )**  
**cpp** YRETCODE **get\_errorType( )**  
**pas** function **get\_errorType( )**: YRETCODE  
**vb** function **get\_errorType( )** As YRETCODE  
**cs** YRETCODE **get\_errorType( )**  
**java** int **get\_errorType( )**  
**py** def **get\_errorType( )**

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

**Returns :**

a number corresponding to the code of the latest error that 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 exemple: 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) <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	<b>def 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>
node.js	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( ) As String</b>
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()** **YGenericSensor**  
**genericsensor→hardwareId()** **genericsensor→get\_hardwareId( )**

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

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the generic sensor. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the generic sensor (ex: RELAYL01-123456.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>
node.js	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( ): double</b>
vb	function <b>get_highestValue( ) As Double</b>
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YGenericSensor <b>target 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.

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

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

**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()**  
**genericsensor→module()genericsensor→**  
**get\_module( )**

**YGenericSensor**

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

**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 get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**genericSensor→get\_recordedData()****YGenericSensor**

**genericSensor→recordedData()** genericSensor→  
**get\_recordedData()**

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

<b>js</b>	function <b>get_recordedData( startTime, endTime)</b>
<b>node.js</b>	function <b>get_recordedData( startTime, endTime)</b>
<b>php</b>	function <b>get_recordedData( \$startTime, \$endTime)</b>
<b>cpp</b>	<b>YDataSet get_recordedData( s64 startTime, s64 endTime)</b>
<b>m</b>	-( <b>YDataSet*</b> ) <b>recordedData</b> : (s64) <b>startTime</b> : (s64) <b>endTime</b>
<b>pas</b>	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
<b>vb</b>	function <b>get_recordedData( ) As YDataSet</b>
<b>cs</b>	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
<b>java</b>	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
<b>py</b>	<b>def get_recordedData( startTime, endTime)</b>
<b>cmd</b>	<b>YGenericSensor target get_recordedData startTime 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 measure, 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 measure, 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()`  
`genericsensor→reportFrequency()``genericsensor→get_reportFrequency( )`

**YGenericSensor**

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

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

**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()**  
**genericsensor→resolution()****genericsensor→get\_resolution( )**

**YGenericSensor**

Returns the resolution of the measured values.

<b>js</b>	function <b>get_resolution( )</b>
<b>nodejs</b>	function <b>get_resolution( )</b>
<b>php</b>	function <b>get_resolution( )</b>
<b>cpp</b>	double <b>get_resolution( )</b>
<b>m</b>	-(double) resolution
<b>pas</b>	function <b>get_resolution( ): double</b>
<b>vb</b>	function <b>get_resolution( ) As Double</b>
<b>cs</b>	double <b>get_resolution( )</b>
<b>java</b>	double <b>get_resolution( )</b>
<b>py</b>	<b>def get_resolution( )</b>
<b>cmd</b>	<b>YGenericSensor target 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 get_signalRange( )
nodejs	function get_signalRange( )
php	function get_signalRange( )
cpp	string get_signalRange( )
m	-(NSString*) signalRange
pas	function get_signalRange( ): string
vb	function get_signalRange( ) As String
cs	string get_signalRange( )
java	String get_signalRange( )
py	def get_signalRange( )
cmd	YGenericSensor target get_signalRange

**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()**  
**genericsensor→signalUnit() genericsensor→**  
**get\_signalUnit( )**

**YGenericSensor**

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*) <b>signalUnit</b>
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 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()**  
**genericsensor→signalValue()****genericsensor→get\_signalValue()**

**YGenericSensor**

Returns the measured value of the electrical signal used by the sensor.

**js** function **get\_signalValue( )**  
**nodejs** function **get\_signalValue( )**  
**php** function **get\_signalValue( )**  
**cpp** double **get\_signalValue( )**  
**m** -(double) **signalValue**  
**pas** function **get\_signalValue( )**: double  
**vb** function **get\_signalValue( )** As Double  
**cs** double **get\_signalValue( )**  
**java** double **get\_signalValue( )**  
**py** def **get\_signalValue( )**  
**cmd** YGenericSensor **target get\_signalValue**

**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*) <b>unit</b>
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	<b>YGenericSensor target 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)`.

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

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()****YGenericSensor****genericsensor→valueRange() genericsensor→  
get\_valueRange( )**

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*) <b>valueRange</b>
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 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>
node.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 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()****YGenericSensor**

Preloads the generic sensor cache with a specified validity duration.

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

Continues the enumeration of generic sensors started using `yFirstGenericSensor()`.

js	<code>function nextGenericSensor( )</code>
nodejs	<code>function nextGenericSensor( )</code>
php	<code>function nextGenericSensor( )</code>
cpp	<code>YGenericSensor * nextGenericSensor( )</code>
m	<code>-(YGenericSensor*) nextGenericSensor</code>
pas	<code>function nextGenericSensor( ): TYGenericSensor</code>
vb	<code>function nextGenericSensor( ) As YGenericSensor</code>
cs	<code>YGenericSensor nextGenericSensor( )</code>
java	<code>YGenericSensor nextGenericSensor( )</code>
py	<code>def nextGenericSensor( )</code>

**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()****YGenericSensor****genericsensor→****registerTimedReportCallback( )**

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

<b>js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>nodejs</b>	function registerTimedReportCallback( <b>callback</b> )
<b>php</b>	function registerTimedReportCallback( <b>\$callback</b> )
<b>cpp</b>	int registerTimedReportCallback( YGenericSensorTimedReportCallback <b>callback</b> )
<b>m</b>	- (int) registerTimedReportCallback : (YGenericSensorTimedReportCallback) <b>callback</b>
<b>pas</b>	function registerTimedReportCallback( <b>callback</b> : TYGenericSensorTimedReportCallback): LongInt
<b>vb</b>	function registerTimedReportCallback( ) As Integer
<b>cs</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>java</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>py</b>	def registerTimedReportCallback( <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	<code>function registerValueCallback( callback)</code>
node.js	<code>function registerValueCallback( callback)</code>
php	<code>function registerValueCallback( \$callback)</code>
cpp	<code>int registerValueCallback( YGenericSensorValueCallback callback)</code>
m	<code>-(int) registerValueCallback : (YGenericSensorValueCallback) callback</code>
pas	<code>function registerValueCallback( callback: TYGenericSensorValueCallback): LongInt</code>
vb	<code>function registerValueCallback( ) As Integer</code>
cs	<code>int registerValueCallback( ValueCallback callback)</code>
java	<code>int registerValueCallback( UpdateCallback callback)</code>
py	<code>def registerValueCallback( callback)</code>

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 set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YGenericSensor target set_highestValue newval
```

**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.

<b>genericsensor→set_logFrequency()</b>	<b>YGenericSensor</b>
<b>genericsensor→setLogFrequency()genericsensor →set_logFrequency( )</b>	

Changes the datalogger recording frequency for this function.

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

**genericsensor→set\_logicalName()** **YGenericSensor**  
**genericsensor→setLogicalName()****genericsensor→set\_logicalName( )**

Changes the logical name of the generic sensor.

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

**Returns :**

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

**genericsensor→set\_lowestValue()** **YGenericSensor**  
**genericsensor→setLowestValue()** genericsensor→  
**set\_lowestValue( )**

Changes the recorded minimal value observed.

```
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 YGenericSensor target set_lowestValue newval
```

**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.

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

<b>genericsensor→set_resolution()</b>	<b>YGenericSensor</b>
<b>genericsensor→setResolution()genericsensor→set_resolution( )</b>	

Changes the resolution of the measured physical values.

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

**genericsensor→set\_signalRange()**  
**genericsensor→setSignalRange()****genericsensor→set\_signalRange( )**

**YGenericSensor**

Changes the electric signal range used by the sensor.

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

**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()**  
**genericsensor→setUnit()** genericsensor→  
**set\_unit( )**

**YGenericSensor**

Changes the measuring unit for the measured value.

<b>js</b>	function <b>set_unit( newval)</b>
<b>nodejs</b>	function <b>set_unit( newval)</b>
<b>php</b>	function <b>set_unit( \$newval)</b>
<b>cpp</b>	int <b>set_unit( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setUnit : (NSString*) newval</b>
<b>pas</b>	function <b>set_unit( newval: string): integer</b>
<b>vb</b>	function <b>set_unit( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_unit( string newval)</b>
<b>java</b>	int <b>set_unit( String newval)</b>
<b>py</b>	def <b>set_unit( newval)</b>
<b>cmd</b>	<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>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b> void* <b>data</b>
m	-(void) <b>setUserData</b> : (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 set_valueRange( newval)
nodejs function set_valueRange( newval)
php  function set_valueRange( $newval)
cpp  int set_valueRange( const string& newval)
m    -(int) setValueRange : (NSString*) newval
pas   function set_valueRange( newval: string): integer
vb   function set_valueRange( ByVal newval As String) As Integer
cs   int set_valueRange( string newval)
java  int set_valueRange( String newval)
py   def set_valueRange( newval)
cmd  YGenericSensor target set_valueRange newval
```

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');
cpp	#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

#### **yocto\_gyro(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.

<b>gyro→get_heading()</b>	Returns the estimated heading angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.
<b>gyro→get_highestValue()</b>	Returns the maximal value observed for the angular velocity since the device was started.
<b>gyro→get_logFrequency()</b>	Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>gyro→get_logicalName()</b>	Returns the logical name of the gyroscope.
<b>gyro→get_lowestValue()</b>	Returns the minimal value observed for the angular velocity since the device was started.
<b>gyro→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>gyro→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>gyro→get_pitch()</b>	Returns the estimated pitch angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.
<b>gyro→get_quaternionW()</b>	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.
<b>gyro→get_quaternionX()</b>	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.
<b>gyro→get_quaternionY()</b>	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.
<b>gyro→get_quaternionZ()</b>	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.
<b>gyro→get_recordedData(startTime, endTime)</b>	Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>gyro→get_reportFrequency()</b>	Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>gyro→get_resolution()</b>	Returns the resolution of the measured values.
<b>gyro→get_roll()</b>	Returns the estimated roll angle, based on the integration of gyroscopic measures combined with acceleration and magnetic field measurements.
<b>gyro→get_unit()</b>	Returns the measuring unit for the angular velocity.
<b>gyro→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>gyro→get_xValue()</b>	Returns the angular velocity around the X axis of the device, as a floating point number.
<b>gyro→get_yValue()</b>	

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()yFindGyro( )

YGyro

Retrieves a gyroscope for a given identifier.

```
js function yFindGyro( func)
node.js function FindGyro( func)
php function yFindGyro( $func)
cpp YGyro* yFindGyro( string func)
m +(YGyro*) yFindGyro : (NSString*) func
pas function yFindGyro( func: string): TYGyro
vb function yFindGyro( ByVal func As String) As YGyro
cs YGyro FindGyro( string func)
java YGyro FindGyro( String func)
def FindGyro( 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 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()yFirstGyro( )

## YGyro

Starts the enumeration of gyroscopes currently accessible.

```
js function yFirstGyro( )
nodejs function FirstGyro( )
php function yFirstGyro( )
cpp YGyro* yFirstGyro( )
m YGyro* yFirstGyro( )
pas function yFirstGyro( ): TYGyro
vb function yFirstGyro( ) As YGyro
cs YGyro FirstGyro( )
java YGyro FirstGyro( )
py def FirstGyro( )
```

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()****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 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)=RELAYL01-123456.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 gyroscope (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**gyro→get\_advertisedValue()**  
**gyro→advertisedValue()** gyro→  
**get\_advertisedValue( )**

**YGyro**

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

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

**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()**  
**gyro→currentRawValue()** gyro→  
**get\_currentRawValue( )**

**YGyro**

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

<b>js</b>	function <b>get_currentRawValue( )</b>
<b>nodejs</b>	function <b>get_currentRawValue( )</b>
<b>php</b>	function <b>get_currentRawValue( )</b>
<b>cpp</b>	double <b>get_currentRawValue( )</b>
<b>m</b>	-(double) currentRawValue
<b>pas</b>	function <b>get_currentRawValue( ): double</b>
<b>vb</b>	function <b>get_currentRawValue( ) As Double</b>
<b>cs</b>	double <b>get_currentRawValue( )</b>
<b>java</b>	double <b>get_currentRawValue( )</b>
<b>py</b>	def <b>get_currentRawValue( )</b>
<b>cmd</b>	<b>YGyro target 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()**  
**gyro→currentValue()****gyro→get\_currentValue()**

**YGyro**

Returns the current value of the angular velocity.

```
js function get_currentValue( )
node.js function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YGyro target get_currentValue
```

**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>getErrorMessage( )</b>
nodejs	function <b>getErrorMessage( )</b>
php	function <b>getErrorMessage( )</b>
cpp	string <b>getErrorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>getErrorMessage( )</b> : string
vb	function <b>getErrorMessage( )</b> As String
cs	string <b>getErrorMessage( )</b>
java	String <b>getErrorMessage( )</b>
py	def <b>getErrorMessage( )</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()**  
**gyro→errorType()gyro→get\_errorType( )****YGyro**

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

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

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

**YGyro**

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

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	String <b>get_functionDescriptor( )</b>
<b>py</b>	<b>def 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( ) As String</b>
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 get_hardwareId( )  
node.js function get_hardwareId( )  
php function get_hardwareId( )  
cpp string get_hardwareId( )  
m -(NSString*) hardwareId  
vb function get_hardwareId( ) As String  
cs string get_hardwareId( )  
java String get_hardwareId( )  
py def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the gyroscope. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the gyroscope (ex: RELAYL01-123456.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 get_heading()
nodejs function get_heading()
php function get_heading()
cpp double get_heading()
m -(double) heading
pas function get_heading(): double
vb function get_heading() As Double
cs double get_heading()
java double get_heading()
py def get_heading()
```

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 get_highestValue( )
node.js function get_highestValue( )
php function get_highestValue( )
cpp double get_highestValue( )
m -(double) highestValue
pas function get_highestValue( ): double
vb function get_highestValue( ) As Double
cs double get_highestValue( )
java double get_highestValue( )
py def get_highestValue( )
cmd YGyro target get_highestValue
```

**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>
node.js	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( ): string</b>
vb	function <b>get_logFrequency( ) As String</b>
cs	string <b>get_logFrequency( )</b>
java	String <b>get_logFrequency( )</b>
py	def <b>get_logFrequency( )</b>
cmd	YGyro <b>target 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>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	- <b>(NSString*) logicalName</b>
pas	function <b>get_logicalName( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	<b>String get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	<b>YGyro target 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 get_lowestValue( )
nodejs function get_lowestValue( )
php function get_lowestValue( )
cpp double get_lowestValue( )
m -(double) lowestValue
pas function get_lowestValue( ): double
vb function get_lowestValue( ) As Double
cs double get_lowestValue( )
java double get_lowestValue( )
py def get_lowestValue( )
cmd YGyro target get_lowestValue
```

**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()**  
**gyro→module()gyro→get\_module( )**

**YGyro**

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

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

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

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

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

### Returns :

nothing : the result is provided to the callback.

**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 get_pitch( )
nodejs function get_pitch( )
php function get_pitch( )
cpp double get_pitch( )
m -(double) pitch
pas function get_pitch( ): double
vb function get_pitch( ) As Double
cs double get_pitch( )
java double get_pitch( )
py def get_pitch( )
```

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 get_quaternionW( )
nodejs function get_quaternionW( )
php function get_quaternionW( )
cpp double get_quaternionW( )
m -(double) quaternionW
pas function get_quaternionW( ): double
vb function get_quaternionW( ) As Double
cs double get_quaternionW( )
java double get_quaternionW( )
py def get_quaternionW( )
```

**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( ): double</b>
vb	function <b>get_quaternionX( ) As Double</b>
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 get_quaternionY( )
nodejs function get_quaternionY( )
php function get_quaternionY( )
cpp double get_quaternionY( )
m -(double) quaternionY
pas function get_quaternionY( ): double
vb function get_quaternionY( ) As Double
cs double get_quaternionY( )
java double get_quaternionY( )
py def get_quaternionY( )
```

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 get_quaternionZ( )
nodejs function get_quaternionZ( )
php function get_quaternionZ( )
cpp double get_quaternionZ( )
m -(double) quaternionZ
pas function get_quaternionZ( ): double
vb function get_quaternionZ( ) As Double
cs double get_quaternionZ( )
java double get_quaternionZ( )
py def get_quaternionZ( )
```

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.

<code>js</code>	<code>function get_recordedData( startTime, endTime)</code>
<code>node.js</code>	<code>function get_recordedData( startTime, endTime)</code>
<code>php</code>	<code>function get_recordedData( \$startTime, \$endTime)</code>
<code>cpp</code>	<code>YDataSet get_recordedData( s64 startTime, s64 endTime)</code>
<code>m</code>	<code>-(YDataSet*) recordedData : (s64) startTime : (s64) endTime</code>
<code>pas</code>	<code>function get_recordedData( startTime: int64, endTime: int64): TYDataSet</code>
<code>vb</code>	<code>function get_recordedData( ) As YDataSet</code>
<code>cs</code>	<code>YDataSet get_recordedData( long startTime, long endTime)</code>
<code>java</code>	<code>YDataSet get_recordedData( long startTime, long endTime)</code>
<code>py</code>	<code>def get_recordedData( startTime, endTime)</code>
<code>cmd</code>	<code>YGyro target get_recordedData startTime endTime</code>

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 measure, 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 measure, 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.

---

<b>gyro→get_reportFrequency()</b>	<b>YGyro</b>
<b>gyro→reportFrequency()</b>	<b>YGyro→get_reportFrequency()</b>

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

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

**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() gyro→resolution()gyro→get\_resolution()

YGyro

Returns the resolution of the measured values.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YGyro target get_resolution
```

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	<b>def 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>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) <b>unit</b>
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 target <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)****YGyro****gyro→userData()gyro→get(userData()**

Returns the value of the userData attribute, as previously stored using method set(userData).

js	function get(userData)
node.js	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): TObject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

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

**Returns :**

the object stored previously by the caller.

**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 **get\_xValue( )****nodejs** function **get\_xValue( )****php** function **get\_xValue( )****cpp** double **get\_xValue( )****m** -(double) **xValue****pas** function **get\_xValue( ): double****vb** function **get\_xValue( ) As Double****cs** double **get\_xValue( )****java** double **get\_xValue( )****py** def **get\_xValue( )****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()**  
**gyro→yValue()gyro→get\_yValue( )****YGyro**

Returns the angular velocity around the Y axis of the device, as a floating point number.

js	function <b>get_yValue( )</b>
node.js	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( ): double</b>
vb	function <b>get_zValue( ) As Double</b>
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()****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	- <b>(BOOL) 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()****YGyro**

Preloads the gyroscope cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-( <b>YRETCODE</b> ) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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()****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> : TGYro
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 registerAnglesCallback( callback)
nodejs function registerAnglesCallback( callback)
php  function registerAnglesCallback( $callback)
cpp   int registerAnglesCallback( YAnglesCallback callback)
m    -(int) registerAnglesCallback : (YAnglesCallback) callback
pas   function registerAnglesCallback( callback: TYAnglesCallback): LongInt
vb    function registerAnglesCallback( ) As Integer
cs   int registerAnglesCallback( YAnglesCallback callback)
java  int registerAnglesCallback( YAnglesCallback callback)
py    def registerAnglesCallback( callback)
```

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 registerQuaternionCallback( <b>callback</b> )
node.js	function registerQuaternionCallback( <b>callback</b> )
php	function registerQuaternionCallback( \$callback)
cpp	int registerQuaternionCallback( YQuatCallback <b>callback</b> )
m	-(int) registerQuaternionCallback : (YQuatCallback) <b>callback</b>
pas	function registerQuaternionCallback( <b>callback</b> : TYQuatCallback): LongInt
vb	function registerQuaternionCallback( ) As Integer
cs	int registerQuaternionCallback( YQuatCallback <b>callback</b> )
java	int registerQuaternionCallback( YQuatCallback <b>callback</b> )
py	def registerQuaternionCallback( <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 registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( \$callback)
cpp	int registerTimedReportCallback( YGyroTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YGyroTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYGyroTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YGyroValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YGyroValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYGyroValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**gyro→setHighestValue()** gyro→  
**set\_highestValue( )**

**YGyro**

Changes the recorded maximal value observed.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YGyro target set_highestValue newval
```

**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()**  
**gyro→setLogFrequency()** gyro→  
**set\_logFrequency( )****YGyro**

Changes the datalogger recording frequency for this function.

<code>js</code>	function <b>set_logFrequency( newval)</b>
<code>nodejs</code>	function <b>set_logFrequency( newval)</b>
<code>php</code>	function <b>set_logFrequency( \$newval)</b>
<code>cpp</code>	int <b>set_logFrequency( const string&amp; newval)</b>
<code>m</code>	-(int) setLogFrequency : (NSString*) <b>newval</b>
<code>pas</code>	function <b>set_logFrequency( newval: string): integer</b>
<code>vb</code>	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
<code>cs</code>	int <b>set_logFrequency( string newval)</b>
<code>java</code>	int <b>set_logFrequency( String newval)</b>
<code>py</code>	def <b>set_logFrequency( newval)</b>
<code>cmd</code>	<b>YGyro 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() gyro→setLogicalName()gyro→set\_logicalName( )

YGyro

Changes the logical name of the gyroscope.

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

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

### Parameters :

**newval** a string corresponding to the logical name of the 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 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 YGyro target set_lowestValue newval
```

**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.

---

<b>gyro→set_reportFrequency()</b>	<b>YGyro</b>
<b>gyro→setReportFrequency()</b> gyro→ <b>set_reportFrequency( )</b>	

---

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   YGyro 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.

**gyro→set\_resolution()****YGyro****gyro→setResolution()gyro→set\_resolution( )**

Changes the resolution of the measured physical values.

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

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

js	function <b>set(userData( data)</b>
node.js	function <b>set(userData( data)</b>
php	function <b>set(userData( \$data)</b>
cpp	void <b>set(userData( void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData( data: Tobject)</b>
vb	procedure <b>set(userData( ByVal data As Object)</b>
cs	void <b>set(userData( object data)</b>
java	void <b>set(userData( Object data)</b>
py	def <b>set(userData( 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');
cpp	#include "yocto_hubport.h"
m	#import "yocto_hubport.h"
pas	uses yocto_hubport;
vb	yocto_hubport.vb
cs	yocto_hubport.cs
java	import com.yoctopuce.YoctoAPI.YHubPort;
py	from yocto_hubport import *

### Global functions

#### yFindHubPort(func)

Retrieves a Yocto-hub port for a given identifier.

#### yFirstHubPort()

Starts the enumeration of Yocto-hub ports currently accessible.

### YHubPort methods

#### hubport→describe()

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

#### hubport→get\_advertisedValue()

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

#### hubport→get\_baudRate()

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

#### hubport→get\_enabled()

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

#### hubport→get\_errorMessage()

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

#### hubport→get\_errorType()

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

#### hubport→get\_friendlyName()

Returns a global identifier of the Yocto-hub port in the format MODULE\_NAME . FUNCTION\_NAME.

#### hubport→get\_functionDescriptor()

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

#### hubport→get\_functionId()

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

#### hubport→get\_hardwareId()

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

#### hubport→get\_logicalName()

Returns the logical name of the Yocto-hub port.

<b>hubport→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>hubport→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>hubport→get_portState()</b>	Returns the current state of the Yocto-hub port.
<b>hubport→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>hubport→isOnline()</b>	Checks if the Yocto-hub port is currently reachable, without raising any error.
<b>hubport→isOnline_async(callback, context)</b>	Checks if the Yocto-hub port is currently reachable, without raising any error (asynchronous version).
<b>hubport→load(msValidity)</b>	Preloads the Yocto-hub port cache with a specified validity duration.
<b>hubport→load_async(msValidity, callback, context)</b>	Preloads the Yocto-hub port cache with a specified validity duration (asynchronous version).
<b>hubport→nextHubPort()</b>	Continues the enumeration of Yocto-hub ports started using yFirstHubPort( ).
<b>hubport→registerValueCallback(callback)</b>	Registers the callback function that is invoked on every change of advertised value.
<b>hubport→set_enabled(newval)</b>	Changes the activation of the Yocto-hub port.
<b>hubport→set_logicalName(newval)</b>	Changes the logical name of the Yocto-hub port.
<b>hubport→set_userData(data)</b>	Stores a user context provided as argument in the userData attribute of the function.
<b>hubport→wait_async(callback, context)</b>	Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YHubPort.FindHubPort() yFindHubPort(yFindHubPort( )

YHubPort

Retrieves a Yocto-hub port for a given identifier.

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

The identifier can be specified using several formats:

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

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

### Parameters :

`func` a string that uniquely characterizes the Yocto-hub port

### Returns :

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

**YHubPort.FirstHubPort()****yFirstHubPort()yFirstHubPort( )****YHubPort**

Starts the enumeration of Yocto-hub ports currently accessible.

js	function <b>yFirstHubPort( )</b>
nodejs	function <b>FirstHubPort( )</b>
php	function <b>yFirstHubPort( )</b>
cpp	YHubPort* <b>yFirstHubPort( )</b>
m	YHubPort* <b>yFirstHubPort( )</b>
pas	function <b>yFirstHubPort( )</b> : TYHubPort
vb	function <b>yFirstHubPort( )</b> As YHubPort
cs	YHubPort <b>FirstHubPort( )</b>
java	YHubPort <b>FirstHubPort( )</b>
py	def <b>FirstHubPort( )</b>

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

**Returns :**

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

**hubport→describe()****YHubPort**

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

js	function <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)=RELAYL01-123456.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 Yocto-hub port (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**hubport→get\_advertisedValue()**  
**hubport→advertisedValue()**  
**hubport→get\_advertisedValue( )**

**YHubPort**

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

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>node.js</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YHubPort target 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>
node.js	function <b>get_baudRate( )</b>
php	function <b>get_baudRate( )</b>
cpp	int <b>get_baudRate( )</b>
m	-(int) <b>baudRate</b>
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 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	<b>Y_ENABLED_enum get_enabled( )</b>
m	-( <b>Y_ENABLED_enum</b> ) 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	<b>YHubPort 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()**  
**hubport→errorMessage()** hubport→  
**get\_errorMessage( )**

**YHubPort**

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

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

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

**Returns :**

a string corresponding to the latest error message that 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.

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 Yocto-hub port object

---

<b>hubport→get_friendlyName()</b>	<b>YHubPort</b>
<b>hubport→friendlyName()hubport→</b>	
<b>get_friendlyName( )</b>	

---

Returns a global identifier of the Yocto-hub port in the format MODULE\_NAME.FUNCTION\_NAME.

<b>js</b>	function <b>get_friendlyName( )</b>
<b>nodejs</b>	function <b>get_friendlyName( )</b>
<b>php</b>	function <b>get_friendlyName( )</b>
<b>cpp</b>	string <b>get_friendlyName( )</b>
<b>m</b>	- <b>(NSString*) friendlyName</b>
<b>cs</b>	string <b>get_friendlyName( )</b>
<b>java</b>	String <b>get_friendlyName( )</b>
<b>py</b>	<b>def 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 exemple: MyCustomName.relay1)

**Returns :**

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

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

**hubport→get\_functionDescriptor()**  
**hubport→functionDescriptor()hubport→**  
**get\_functionDescriptor( )**

**YHubPort**

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

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

---

<b>hubport→get_hardwareId()</b>	<b>YHubPort</b>
<b>hubport→hardwareId()</b> hubport→	
<b>get_hardwareId( )</b>	

---

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

<b>js</b>	function <b>get_hardwareId( )</b>
<b>node.js</b>	function <b>get_hardwareId( )</b>
<b>php</b>	function <b>get_hardwareId( )</b>
<b>cpp</b>	string <b>get_hardwareId( )</b>
<b>m</b>	-(NSString*) hardwareId
<b>vb</b>	function <b>get_hardwareId( )</b> As String
<b>cs</b>	string <b>get_hardwareId( )</b>
<b>java</b>	String <b>get_hardwareId( )</b>
<b>py</b>	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the Yocto-hub port (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**hubport→get\_logicalName()**  
**hubport→logicalName()** hubport→  
**get\_logicalName( )**

**YHubPort**

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 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>
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 online.

**Returns :**

an instance of YModule

**hubport→get\_module\_async()**  
**hubport→module\_async()****YHubPort**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

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

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

**Returns :**

the object stored previously by the caller.

**hubport→isOnline()****YHubPort**

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

js	function <b>isOnline</b> ( )
node.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 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.

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

## 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()****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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YHubPortValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YHubPortValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYHubPortValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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.

---

<b>hubport→set_enabled()</b>	<b>YHubPort</b>
<b>hubport→setEnabled()hubport→set_enabled()</b>	

---

Changes the activation of the Yocto-hub port.

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

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()**  
**hubport→setLogicalName()** hubport→  
**set\_logicalName( )**

**YHubPort**

Changes the logical name of the Yocto-hub port.

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

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

**Parameters :**

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

**Returns :**

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

**hubport→set(userData)****YHubPort****hubport→setUserData()hubport→set(userData())**

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

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

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

**Parameters :**

**data** any kind of object to be stored

## hubport→wait\_async()

YHubPort

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

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

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

### Parameters :

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

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

### Returns :

nothing.

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

### 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.

### 3. Reference

<b>humidity→get_highestValue()</b>
Returns the maximal value observed for the humidity.
<b>humidity→get_logFrequency()</b>
Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>humidity→get_logicalName()</b>
Returns the logical name of the humidity sensor.
<b>humidity→get_lowestValue()</b>
Returns the minimal value observed for the humidity.
<b>humidity→get_module()</b>
Gets the YModule object for the device on which the function is located.
<b>humidity→get_module_async(callback, context)</b>
Gets the YModule object for the device on which the function is located (asynchronous version).
<b>humidity→get_recordedData(startTime, endTime)</b>
Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>humidity→get_reportFrequency()</b>
Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>humidity→get_resolution()</b>
Returns the resolution of the measured values.
<b>humidity→get_unit()</b>
Returns the measuring unit for the humidity.
<b>humidity→get(userData)</b>
Returns the value of the userData attribute, as previously stored using method set(userData).
<b>humidity→isOnline()</b>
Checks if the humidity sensor is currently reachable, without raising any error.
<b>humidity→isOnline_async(callback, context)</b>
Checks if the humidity sensor is currently reachable, without raising any error (asynchronous version).
<b>humidity→load(msValidity)</b>
Preloads the humidity sensor cache with a specified validity duration.
<b>humidity→loadCalibrationPoints(rawValues, refValues)</b>
Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>humidity→load_async(msValidity, callback, context)</b>
Preloads the humidity sensor cache with a specified validity duration (asynchronous version).
<b>humidity→nextHumidity()</b>
Continues the enumeration of humidity sensors started using yFirstHumidity().
<b>humidity→registerTimedReportCallback(callback)</b>
Registers the callback function that is invoked on every periodic timed notification.
<b>humidity→registerValueCallback(callback)</b>
Registers the callback function that is invoked on every change of advertised value.
<b>humidity→set_highestValue(newval)</b>
Changes the recorded maximal value observed for the humidity.
<b>humidity→set_logFrequency(newval)</b>
Changes the datalogger recording frequency for this function.
<b>humidity→set_logicalName(newval)</b>
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()****YHumidity****yFindHumidity()yFindHumidity( )**

Retrieves a humidity sensor for a given identifier.

```
js function yFindHumidity( func)
node.js function FindHumidity( func)
php function yFindHumidity( $func)
cpp YHumidity* yFindHumidity( const string& func)
m YHumidity* yFindHumidity( NSString* func)
pas function yFindHumidity( func: string): TYHumidity
vb function yFindHumidity( ByVal func As String) As YHumidity
cs YHumidity FindHumidity( string func)
java YHumidity FindHumidity( String func)
def FindHumidity( 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 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()yFirstHumidity()

## 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()****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)=RELAYL01-123456.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 humidity sensor (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**humidity→get\_advertisedValue()**  
**humidity→advertisedValue()**humidity→  
**get\_advertisedValue( )**

YHumidity

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

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

**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()**  
**humidity→currentRawValue()** **humidity→get\_currentRawValue( )**

**YHumidity**

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

<b>js</b>	function <b>get_currentRawValue( )</b>
<b>nodejs</b>	function <b>get_currentRawValue( )</b>
<b>php</b>	function <b>get_currentRawValue( )</b>
<b>cpp</b>	double <b>get_currentRawValue( )</b>
<b>m</b>	-(double) currentRawValue
<b>pas</b>	function <b>get_currentRawValue( ): double</b>
<b>vb</b>	function <b>get_currentRawValue( ) As Double</b>
<b>cs</b>	double <b>get_currentRawValue( )</b>
<b>java</b>	double <b>get_currentRawValue( )</b>
<b>py</b>	def <b>get_currentRawValue( )</b>
<b>cmd</b>	YHumidity <b>target 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()**  
**humidity→currentValue()** **humidity→get\_currentValue( )**

**YHumidity**

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) <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	<b>YHumidity target 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→getErrorMessage()  
humidity→errorMessage()humidity→  
getErrorMessage()****YHumidity**

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

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

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>
node.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 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>
node.js	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 exemple: 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.

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	String <b>get_functionDescriptor( )</b>
<b>py</b>	<b>def 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()**  
**humidity→functionId()** **humidity→get\_functionId( )**

**YHumidity**

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

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the humidity sensor. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the humidity sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**humidity→get\_highestValue()**  
**humidity→highestValue()** **humidity→**  
**get\_highestValue()**

**YHumidity**

Returns the maximal value observed for the humidity.

<b>js</b>	function <b>get_highestValue( )</b>
<b>nodejs</b>	function <b>get_highestValue( )</b>
<b>php</b>	function <b>get_highestValue( )</b>
<b>cpp</b>	double <b>get_highestValue( )</b>
<b>m</b>	-(double) <b>highestValue</b>
<b>pas</b>	function <b>get_highestValue( )</b> : double
<b>vb</b>	function <b>get_highestValue( )</b> As Double
<b>cs</b>	double <b>get_highestValue( )</b>
<b>java</b>	double <b>get_highestValue( )</b>
<b>py</b>	def <b>get_highestValue( )</b>
<b>cmd</b>	<b>YHumidity target 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()**  
**humidity→logFrequency()** **humidity→get\_logFrequency( )**

**YHumidity**

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

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

**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()**  
**humidity→logicalName()**humidity→  
**get\_logicalName( )**

**YHumidity**

Returns the logical name of the humidity sensor.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	YHumidity <b>target 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()**  
**humidity→lowestValue()** **humidity→get\_lowestValue()**

**YHumidity**

Returns the minimal value observed for the humidity.

```
js function get_lowestValue( )  
nodejs function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YHumidity target get_lowestValue
```

**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>
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 online.

**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 get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**humidity→get\_recordedData()**  
**humidity→recordedData()humidity→**  
**get\_recordedData()**

**YHumidity**

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

<b>js</b>	function <b>get_recordedData( startTime, endTime)</b>
<b>node.js</b>	function <b>get_recordedData( startTime, endTime)</b>
<b>php</b>	function <b>get_recordedData( \$startTime, \$endTime)</b>
<b>cpp</b>	<b>YDataSet get_recordedData( s64 startTime, s64 endTime)</b>
<b>m</b>	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
<b>pas</b>	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
<b>vb</b>	function <b>get_recordedData( ) As YDataSet</b>
<b>cs</b>	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
<b>java</b>	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
<b>py</b>	<b>def get_recordedData( startTime, endTime)</b>
<b>cmd</b>	<b>YHumidity target get_recordedData startTime 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 measure, 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 measure, 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()**  
**humidity→reportFrequency()**humidity→  
**get\_reportFrequency( )**

YHumidity

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

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

**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()**  
**humidity→resolution()** **humidity→get\_resolution( )**

**YHumidity**

Returns the resolution of the measured values.

<b>js</b>	function <b>get_resolution( )</b>
<b>nodejs</b>	function <b>get_resolution( )</b>
<b>php</b>	function <b>get_resolution( )</b>
<b>cpp</b>	double <b>get_resolution( )</b>
<b>m</b>	-(double) resolution
<b>pas</b>	function <b>get_resolution( ): double</b>
<b>vb</b>	function <b>get_resolution( ) As Double</b>
<b>cs</b>	double <b>get_resolution( )</b>
<b>java</b>	double <b>get_resolution( )</b>
<b>py</b>	<b>def get_resolution( )</b>
<b>cmd</b>	YHumidity <b>target 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>
node.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( ): string</b>
vb	function <b>get_unit( ) As String</b>
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YHumidity target 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	<code>function get(userData) </code>
nodejs	<code>function get(userData) </code>
php	<code>function get(userData) </code>
cpp	<code>void * get(userData) </code>
m	<code>-(void*) userData </code>
pas	<code>function get(userData): Tobject </code>
vb	<code>function get(userData) As Object </code>
cs	<code>object get(userData) </code>
java	<code>Object get(userData) </code>
py	<code>def get(userData) </code>

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()**

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	- <b>(BOOL) 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	<b>def 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()****YHumidity**

Preloads the humidity sensor cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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 registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( \$callback)
cpp	int registerTimedReportCallback( YHumidityTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YHumidityTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYHumidityTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <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.

<code>js</code>	<code>function registerValueCallback( callback)</code>
<code>node.js</code>	<code>function registerValueCallback( callback)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>int registerValueCallback( YHumidityValueCallback callback)</code>
<code>m</code>	<code>-(int) registerValueCallback : (YHumidityValueCallback) callback</code>
<code>pas</code>	<code>function registerValueCallback( callback: TYHumidityValueCallback): LongInt</code>
<code>vb</code>	<code>function registerValueCallback( ) As Integer</code>
<code>cs</code>	<code>int registerValueCallback( ValueCallback callback)</code>
<code>java</code>	<code>int registerValueCallback( UpdateCallback callback)</code>
<code>py</code>	<code>def registerValueCallback( callback)</code>

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()**  
**humidity→setHighestValue()** **humidity→set\_highestValue( )**

**YHumidity**

Changes the recorded maximal value observed for the humidity.

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

**YHumidity**

Changes the datalogger recording frequency for this function.

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

**humidity→set\_logicalName()**  
**humidity→setLogicalName()** **humidity→set\_logicalName( )****YHumidity**

Changes the logical name of the humidity sensor.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the humidity sensor.

**Returns :**

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

**humidity→set\_lowestValue()**  
**humidity→setLowestValue()** **humidity→set\_lowestValue( )**

**YHumidity**

Changes the recorded minimal value observed for the humidity.

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

**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 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   YHumidity 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.

**humidity→set\_resolution()**  
**humidity→setResolution()** **humidity→**  
**set\_resolution( )**

**YHumidity**

Changes the resolution of the measured physical values.

<b>js</b>	function <b>set_resolution( newval)</b>
<b>nodejs</b>	function <b>set_resolution( newval)</b>
<b>php</b>	function <b>set_resolution( \$newval)</b>
<b>cpp</b>	int <b>set_resolution( double newval)</b>
<b>m</b>	-(int) <b>setResolution : (double) newval</b>
<b>pas</b>	function <b>set_resolution( newval: double): integer</b>
<b>vb</b>	function <b>set_resolution( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_resolution( double newval)</b>
<b>java</b>	int <b>set_resolution( double newval)</b>
<b>py</b>	def <b>set_resolution( newval)</b>
<b>cmd</b>	<b>YHumidity 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)**  
**humidity→setUserData()** **humidity→set(userData)**

**YHumidity**

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

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b>
m	-(void) <b>set(userData)</b> : (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> ( <b>object</b> <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');
cpp	#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() yFindLed()yFindLed( )

YLed

Retrieves a led for a given identifier.

```
js function yFindLed( func)
node.js function FindLed( func)
php function yFindLed( $func)
cpp YLed* yFindLed( const string& func)
m YLed* yFindLed( NSString* func)
pas function yFindLed( func: string): TYLed
vb function yFindLed( ByVal func As String) As YLed
cs YLed FindLed( string func)
java YLed FindLed( String func)
py def FindLed( 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 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()****YLed****yFirstLed()yFirstLed( )**

Starts the enumeration of leds currently accessible.

js	function <b>yFirstLed( )</b>
nodejs	function <b>FirstLed( )</b>
php	function <b>yFirstLed( )</b>
cpp	<b>YLed*</b> <b>yFirstLed( )</b>
m	<b>YLed*</b> <b>yFirstLed( )</b>
pas	function <b>yFirstLed( )</b> : TYLed
vb	function <b>yFirstLed( )</b> As <b>YLed</b>
cs	<b>YLed</b> <b>FirstLed( )</b>
java	<b>YLed</b> <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()**

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)=RELAYL01-123456.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 led (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**led→get\_advertisedValue()**  
**led→advertisedValue()**  
**led→get\_advertisedValue( )**

**YLed**

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

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

**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()**  
**led→blinking()led→get\_blinking( )**

YLed

Returns the current led signaling mode.

js	function <b>get_blinking( )</b>
node.js	function <b>get_blinking( )</b>
php	function <b>get_blinking( )</b>
cpp	Y_BLINKING_enum <b>get_blinking( )</b>
m	-(Y_BLINKING_enum) <b>blinking</b>
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 target <b>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()**  
**led->errorType()led->get\_errorType( )**

YLed

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

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

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

**Returns :**

a number corresponding to the code of the latest error that 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	<b>def 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 exemple: 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()**  
**led→functionDescriptor()led→**  
**get\_functionDescriptor( )**

YLed

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the led. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the led (ex: RELAYL01-123456.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 target <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() led->luminosity() led->get\_luminosity()

YLed

Returns the current led intensity (in per cent).

```
js function get_luminosity( )
node.js function get_luminosity( )
php function get_luminosity( )
cpp int get_luminosity( )
m -(int) luminosity
pas function get_luminosity( ): LongInt
vb function get_luminosity( ) As Integer
cs int get_luminosity( )
java int get_luminosity( )
py def get_luminosity( )
cmd YLed target get_luminosity
```

**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 online.

**Returns :**

an instance of YModule

## led→get\_module\_async() led→module\_async()

YLed

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

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

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

### Returns :

nothing : the result is provided to the callback.

## led->get\_power() led->power()led->get\_power( )

YLed

Returns the current led state.

```
js function get_power( )
nodejs function get_power( )
php function get_power( )
cpp Y_POWER_enum get_power( )
m -(Y_POWER_enum) power
pas function get_power( ): Integer
vb function get_power( ) As Integer
cs int get_power( )
java int get_power( )
py def get_power( )
cmd YLed target get_power
```

**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)**  
**led→userData()led→get(userData)**

YLed

Returns the value of the userData attribute, as previously stored using method set(userData).

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

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

**Returns :**

the object stored previously by the caller.

**led→isOnline()****YLed**

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

js	function <b>isOnline( )</b>
node.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	<b>def 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()****YLed**

Preloads the led cache with a specified validity duration.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</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.

## 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()`.

<code>js</code>	<code>function nextLed( )</code>
<code>nodejs</code>	<code>function nextLed( )</code>
<code>php</code>	<code>function nextLed( )</code>
<code>cpp</code>	<code>YLed * nextLed( )</code>
<code>m</code>	<code>-(YLed*) nextLed</code>
<code>pas</code>	<code>function nextLed( ): TYLed</code>
<code>vb</code>	<code>function nextLed( ) As YLed</code>
<code>cs</code>	<code>YLed nextLed( )</code>
<code>java</code>	<code>YLed nextLed( )</code>
<code>py</code>	<code>def nextLed( )</code>

**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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YLedValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YLedValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYLedValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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() led->setBlinking() led->set\_blinking()

YLed

Changes the current led signaling mode.

<code>js</code>	<code>function set_blinking( newval)</code>
<code>nodejs</code>	<code>function set_blinking( newval)</code>
<code>php</code>	<code>function set_blinking( \$newval)</code>
<code>cpp</code>	<code>int set_blinking( Y_BLINKING_enum newval)</code>
<code>m</code>	<code>-(int) setBlinking : (Y_BLINKING_enum) newval</code>
<code>pas</code>	<code>function set_blinking( newval: Integer): integer</code>
<code>vb</code>	<code>function set_blinking( ByVal newval As Integer) As Integer</code>
<code>cs</code>	<code>int set_blinking( int newval)</code>
<code>java</code>	<code>int set_blinking( int newval)</code>
<code>py</code>	<code>def set_blinking( newval)</code>
<code>cmd</code>	<code>YLed target set_blinking newval</code>

### 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()**  
**led->setLogicalName()****led->set\_logicalName( )**

YLed

Changes the logical name of the led.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the 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( newval)</b>
nodejs	function <b>set_luminosity( newval)</b>
php	function <b>set_luminosity( \$newval)</b>
cpp	int <b>set_luminosity( int newval)</b>
m	-(int) setLuminosity : (int) <b>newval</b>
pas	function <b>set_luminosity( newval: LongInt): integer</b>
vb	function <b>set_luminosity( ByVal newval As Integer) As Integer</b>
cs	int <b>set_luminosity( int newval)</b>
java	int <b>set_luminosity( int newval)</b>
py	def <b>set_luminosity( newval)</b>
cmd	YLed target <b>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()**  
**led->setPower()**  
**led->set\_power( )**

YLed

Changes the state of the led.

js	function set_power( newval)
node.js	function set_power( newval)
php	function set_power( \$newval)
cpp	int set_power( Y_POWER_enum newval)
m	-(int) setPower : (Y_POWER_enum) newval
pas	function set_power( newval: Integer): integer
vb	function set_power( ByVal newval As Integer) As Integer
cs	int set_power( int newval)
java	int set_power( int newval)
py	def set_power( newval)
cmd	YLed target set_power newval

**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( data)</b>
node.js	function <b>set(userData( data)</b>
php	function <b>set(userData( \$data)</b>
cpp	void <b>set(userData( void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData( data: Tobject)</b>
vb	procedure <b>set(userData( ByVal data As Object)</b>
cs	void <b>set(userData( object data)</b>
java	void <b>set(userData( Object data)</b>
py	def <b>set(userData( 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>
node.js	var yoctolib = require('yoctolib');
php	var YLightSensor = yoctolib.YLightSensor;
cpp	require_once('yocto_lightsensor.php');
m	#include "yocto_lightsensor.h"
pas	#import "yocto_lightsensor.h"
vb	uses yocto_lightsensor;
cs	yocto_lightsensor.vb
java	yocto_lightsensor.cs
py	import com.yoctopuce.YoctoAPI.YLightSensor;
	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.
<b>lightsensor→get_hardwareId()</b>
Returns the unique hardware identifier of the light sensor in the form SERIAL.FUNCTIONID.
<b>lightsensor→get_highestValue()</b>
Returns the maximal value observed for the ambient light.
<b>lightsensor→get_logFrequency()</b>
Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>lightsensor→get_logicalName()</b>
Returns the logical name of the light sensor.
<b>lightsensor→get_lowestValue()</b>
Returns the minimal value observed for the ambient light.
<b>lightsensor→get_module()</b>
Gets the YModule object for the device on which the function is located.
<b>lightsensor→get_module_async(callback, context)</b>
Gets the YModule object for the device on which the function is located (asynchronous version).
<b>lightsensor→get_recordedData(startTime, endTime)</b>
Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>lightsensor→get_reportFrequency()</b>
Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>lightsensor→get_resolution()</b>
Returns the resolution of the measured values.
<b>lightsensor→get_unit()</b>
Returns the measuring unit for the ambient light.
<b>lightsensor→get(userData)</b>
Returns the value of the userData attribute, as previously stored using method set(userData).
<b>lightsensor→isOnline()</b>
Checks if the light sensor is currently reachable, without raising any error.
<b>lightsensor→isOnline_async(callback, context)</b>
Checks if the light sensor is currently reachable, without raising any error (asynchronous version).
<b>lightsensor→load(msValidity)</b>
Preloads the light sensor cache with a specified validity duration.
<b>lightsensor→loadCalibrationPoints(rawValues, refValues)</b>
Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>lightsensor→load_async(msValidity, callback, context)</b>
Preloads the light sensor cache with a specified validity duration (asynchronous version).
<b>lightsensor→nextLightSensor()</b>
Continues the enumeration of light sensors started using yFirstLightSensor( ).
<b>lightsensor→registerTimedReportCallback(callback)</b>
Registers the callback function that is invoked on every periodic timed notification.
<b>lightsensor→registerValueCallback(callback)</b>
Registers the callback function that is invoked on every change of advertised value.
<b>lightsensor→set_highestValue(newval)</b>
Changes the recorded maximal value observed for the ambient light.
<b>lightsensor→set_logFrequency(newval)</b>

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() yFindLightSensor()yFindLightSensor( )

**YLightSensor**

Retrieves a light sensor for a given identifier.

js	function <b>yFindLightSensor( func)</b>
node.js	function <b>FindLightSensor( func)</b>
php	function <b>yFindLightSensor( \$func)</b>
cpp	YLightSensor* <b>yFindLightSensor( const string&amp; func)</b>
m	YLightSensor* <b>yFindLightSensor( NSString* func)</b>
pas	function <b>yFindLightSensor( func: string): TYLightSensor</b>
vb	function <b>yFindLightSensor( ByVal func As String) As YLightSensor</b>
cs	YLightSensor <b>FindLightSensor( string func)</b>
java	YLightSensor <b>FindLightSensor( String func)</b>
py	def <b>FindLightSensor( 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()****YLightSensor****yFirstLightSensor()yFirstLightSensor( )**

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()****YLightSensor**

Changes the sensor-specific calibration parameter so that the current value matches a desired target (linear scaling).

js	function calibrate( calibratedVal)
nodejs	function calibrate( calibratedVal)
php	function calibrate( \$calibratedVal)
cpp	int calibrate( double calibratedVal)
m	-(int) calibrate : (double) calibratedVal
pas	function calibrate( calibratedVal: double): integer
vb	function calibrate( ByVal calibratedVal As Double) As Integer
cs	int calibrate( double calibratedVal)
java	int calibrate( double calibratedVal)
py	def calibrate( calibratedVal)
cmd	YLightSensor target calibrate calibratedVal

**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 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  YLightSensor 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.

**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 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**lightsensor→get\_advertisedValue()**  
**lightsensor→advertisedValue()****lightsensor→get\_advertisedValue( )**

**YLightSensor**

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

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YLightSensor target 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()**  
**lightsensor→currentRawValue()****lightsensor→get\_currentRawValue( )**

**YLightSensor**

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

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( )**: double  
**vb** function **get\_currentRawValue( )** As Double  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YLightSensor target **get\_currentRawValue**

**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()**  
**lightsensor→currentValue()** lightsensor→  
**get\_currentValue()**

**YLightSensor**

Returns the current measure for the ambient light.

js	function <b>get_currentValue( )</b>
node.js	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	YLightSensor <b>target 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→getErrorMessage()****YLightSensor****lightsensor→errorMessage()**  
**lightsensor→getErrorMessage()**

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

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

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.

<b>js</b>	function <b>get_errorType( )</b>
<b>node.js</b>	function <b>get_errorType( )</b>
<b>php</b>	function <b>get_errorType( )</b>
<b>cpp</b>	<b>YRETCODE get_errorType( )</b>
<b>pas</b>	function <b>get_errorType( ): YRETCODE</b>
<b>vb</b>	function <b>get_errorType( ) As YRETCODE</b>
<b>cs</b>	<b>YRETCODE get_errorType( )</b>
<b>java</b>	<b>int get_errorType( )</b>
<b>py</b>	<b>def 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 light sensor object

**lightsensor→get\_friendlyName()**  
**lightsensor→friendlyName()****lightsensor→get\_friendlyName( )**

**YLightSensor**

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

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

The returned string uses the logical names of the module and of the 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()** YLightSensor  
**lightsensor→functionDescriptor() lightsensor→**  
**get\_functionDescriptor( )**

---

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

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

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()**  
**lightsensor→functionId()** lightsensor→  
**get\_functionId( )**

**YLightSensor**

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*) 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 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>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the light sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**lightsensor→get\_highestValue()**  
**lightsensor→highestValue()****lightsensor→get\_highestValue( )**

**YLightSensor**

Returns the maximal value observed for the ambient light.

**js** function **get\_highestValue( )**  
**nodejs** function **get\_highestValue( )**  
**php** function **get\_highestValue( )**  
**cpp** double **get\_highestValue( )**  
**m** -(double) highestValue  
**pas** function **get\_highestValue( )**: double  
**vb** function **get\_highestValue( )** As Double  
**cs** double **get\_highestValue( )**  
**java** double **get\_highestValue( )**  
**py** def **get\_highestValue( )**  
**cmd** YLightSensor target **get\_highestValue**

**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()**  
**lightsensor→logFrequency()** lightsensor→  
**get\_logFrequency( )**

**YLightSensor**

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

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

**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()**  
**lightsensor→logicalName()** lightsensor→  
**get\_logicalName( )**

**YLightSensor**

Returns the logical name of the light sensor.

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

**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()**  
**lightsensor→lowestValue()****lightsensor→get\_lowestValue( )**

**YLightSensor**

Returns the minimal value observed for the ambient light.

<b>js</b>	function <b>get_lowestValue( )</b>
<b>nodejs</b>	function <b>get_lowestValue( )</b>
<b>php</b>	function <b>get_lowestValue( )</b>
<b>cpp</b>	double <b>get_lowestValue( )</b>
<b>m</b>	-(double) lowestValue
<b>pas</b>	function <b>get_lowestValue( ): double</b>
<b>vb</b>	function <b>get_lowestValue( ) As Double</b>
<b>cs</b>	double <b>get_lowestValue( )</b>
<b>java</b>	double <b>get_lowestValue( )</b>
<b>py</b>	def <b>get_lowestValue( )</b>
<b>cmd</b>	<b>YLightSensor target 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()**  
**lightsensor→module()** lightsensor→  
**get\_module( )**

**YLightSensor**

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

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

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

**Returns :**

an instance of **YModule**

## lightsensor→get\_module\_async() lightsensor→module\_async()

YLightSensor

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

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

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

### Returns :

nothing : the result is provided to the callback.

**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 get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php function get_recordedData( $startTime, $endTime)
cpp YDataSet get_recordedData( s64 startTime, s64 endTime)
m -(YDataSet*) recordedData : (s64) startTime
                           : (s64) endTime
pas function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb function get_recordedData( ) As YDataSet
cs YDataSet get_recordedData( long startTime, long endTime)
java YDataSet get_recordedData( long startTime, long endTime)
py def get_recordedData( startTime, endTime)
cmd YLightSensor target get_recordedData startTime endTime
```

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 measurement, 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 measure, 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()**  
**lightsensor→reportFrequency()****lightsensor→get\_reportFrequency( )**

**YLightSensor**

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

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

**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()**  
**lightsensor→resolution() lightsensor→**  
**get\_resolution()**

**YLightSensor**

Returns the resolution of the measured values.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YLightSensor target get_resolution
```

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*) <b>unit</b>
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 target <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)**  
**lightsensor→userData()** lightsensor→  
**get(userData)**

**YLightSensor**

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()****YLightSensor**

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

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) 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	<b>def 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()****YLightSensor**

Preloads the light sensor cache with a specified validity duration.

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

**lightsensor→loadCalibrationPoints()** lightsensor→  
**loadCalibrationPoints()**

**YLightSensor**

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   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()`.

<code>js</code>	<code>function nextLightSensor( )</code>
<code>node.js</code>	<code>function nextLightSensor( )</code>
<code>php</code>	<code>function nextLightSensor( )</code>
<code>cpp</code>	<code>YLightSensor * nextLightSensor( )</code>
<code>m</code>	<code>-(YLightSensor*) nextLightSensor</code>
<code>pas</code>	<code>function nextLightSensor( ): TYLightSensor</code>
<code>vb</code>	<code>function nextLightSensor( ) As YLightSensor</code>
<code>cs</code>	<code>YLightSensor nextLightSensor( )</code>
<code>java</code>	<code>YLightSensor nextLightSensor( )</code>
<code>py</code>	<code>def nextLightSensor( )</code>

**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()****YLightSensor****lightsensor→registerTimedReportCallback( )**

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

<b>js</b>	<code>function registerTimedReportCallback( callback)</code>
<b>node.js</b>	<code>function registerTimedReportCallback( callback)</code>
<b>php</b>	<code>function registerTimedReportCallback( \$callback)</code>
<b>cpp</b>	<code>int registerTimedReportCallback( YLightSensorTimedReportCallback callback)</code>
<b>m</b>	<code>-(int) registerTimedReportCallback : (YLightSensorTimedReportCallback) callback</code>
<b>pas</b>	<code>function registerTimedReportCallback( callback: TYLightSensorTimedReportCallback): LongInt</code>
<b>vb</b>	<code>function registerTimedReportCallback( ) As Integer</code>
<b>cs</b>	<code>int registerTimedReportCallback( TimedReportCallback callback)</code>
<b>java</b>	<code>int registerTimedReportCallback( TimedReportCallback callback)</code>
<b>py</b>	<code>def registerTimedReportCallback( callback)</code>

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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YLightSensorValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YLightSensorValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYLightSensorValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**lightsensor→setHighestValue()** lightsensor→  
**set\_highestValue()**

**YLightSensor**

Changes the recorded maximal value observed for the ambient light.

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

**YLightSensor**

Changes the datalogger recording frequency for this function.

```
js   function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php  function set_logFrequency( $newval)
cpp   int set_logFrequency( const string& newval)
m    -(int) setLogFrequency : (NSString*) newval
pas   function set_logFrequency( newval: string): integer
vb    function set_logFrequency( ByVal newval As String) As Integer
cs    int set_logFrequency( string newval)
java  int set_logFrequency( String newval)
py    def set_logFrequency( newval)
cmd   YLightSensor target set_logFrequency 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 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()**  
**lightsensor→setLogicalName()**  
**lightsensor→set\_logicalName( )**

**YLightSensor**

Changes the logical name of the light sensor.

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

**Returns :**

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

**lightsensor→set\_lowestValue()**  
**lightsensor→setLowestValue()** lightsensor→  
**set\_lowestValue()**

**YLightSensor**

Changes the recorded minimal value observed for the ambient light.

```
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 YLightSensor target set_lowestValue newval
```

**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()****YLightSensor****lightsensor→setReportFrequency() lightsensor→  
set\_reportFrequency( )**

Changes the timed value notification frequency for this function.

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

**lightsensor→set\_resolution()**  
**lightsensor→setResolution()****lightsensor→set\_resolution()**

**YLightSensor**

Changes the resolution of the measured physical values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YLightSensor target set_resolution newval
```

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)**  
**lightsensor→setUserData()** lightsensor→  
**set(userData)**

**YLightSensor**

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

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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	<script type='text/javascript' src='yocto_magnetometer.js'></script>
nodejs	var yoctolib = require('yoctolib');
php	var YMagnetometer = yoctolib.YMagnetometer;
cpp	require_once('yocto_magnetometer.php');
m	#include "yocto_magnetometer.h"
pas	#import "yocto_magnetometer.h"
vb	uses yocto_magnetometer;
cs	yocto_magnetometer.vb
java	yocto_magnetometer.cs
py	import com.yoctopuce.YoctoAPI.YMagnetometer;
	from yocto_magnetometer import *

### 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.

<b>magnetometer→get_highestValue()</b>	Returns the maximal value observed for the magnetic field since the device was started.
<b>magnetometer→get_logFrequency()</b>	Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>magnetometer→get_logicalName()</b>	Returns the logical name of the magnetometer.
<b>magnetometer→get_lowestValue()</b>	Returns the minimal value observed for the magnetic field since the device was started.
<b>magnetometer→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>magnetometer→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>magnetometer→get_recordedData(startTime, endTime)</b>	Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>magnetometer→get_reportFrequency()</b>	Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>magnetometer→get_resolution()</b>	Returns the resolution of the measured values.
<b>magnetometer→get_unit()</b>	Returns the measuring unit for the magnetic field.
<b>magnetometer→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>magnetometer→get_xValue()</b>	Returns the X component of the magnetic field, as a floating point number.
<b>magnetometer→get_yValue()</b>	Returns the Y component of the magnetic field, as a floating point number.
<b>magnetometer→get_zValue()</b>	Returns the Z component of the magnetic field, as a floating point number.
<b>magnetometer→isOnline()</b>	Checks if the magnetometer is currently reachable, without raising any error.
<b>magnetometer→isOnline_async(callback, context)</b>	Checks if the magnetometer is currently reachable, without raising any error (asynchronous version).
<b>magnetometer→load(msValidity)</b>	Preloads the magnetometer cache with a specified validity duration.
<b>magnetometer→loadCalibrationPoints(rawValues, refValues)</b>	Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>magnetometer→load_async(msValidity, callback, context)</b>	Preloads the magnetometer cache with a specified validity duration (asynchronous version).
<b>magnetometer→nextMagnetometer()</b>	Continues the enumeration of magnetometers started using yFirstMagnetometer( ).
<b>magnetometer→registerTimedReportCallback(callback)</b>	Registers the callback function that is invoked on every periodic timed notification.
<b>magnetometer→registerValueCallback(callback)</b>	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.

**Y Magnetometer.FindMagnetometer()****Y Magnetometer****yFindMagnetometer() yFindMagnetometer( )**

Retrieves a magnetometer for a given identifier.

<b>js</b>	function <b>yFindMagnetometer( func)</b>
<b>node.js</b>	function <b>FindMagnetometer( func)</b>
<b>php</b>	function <b>yFindMagnetometer( \$func)</b>
<b>cpp</b>	Y Magnetometer* <b>yFindMagnetometer( const string&amp; func)</b>
<b>m</b>	Y Magnetometer* <b>yFindMagnetometer( NSString* func)</b>
<b>pas</b>	function <b>yFindMagnetometer( func: string): TYMagnetometer</b>
<b>vb</b>	function <b>yFindMagnetometer( ByVal func As String) As YMagnetometer</b>
<b>cs</b>	Y Magnetometer <b>FindMagnetometer( string func)</b>
<b>java</b>	Y Magnetometer <b>FindMagnetometer( String func)</b>
<b>py</b>	def <b>FindMagnetometer( 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 `Y Magnetometer.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 `Y Magnetometer` object allowing you to drive the magnetometer.

**YMagnetometer.FirstMagnetometer()****YMagnetometer****yFirstMagnetometer()yFirstMagnetometer( )**

Starts the enumeration of magnetometers currently accessible.

```
js function yFirstMagnetometer( )
nodejs function FirstMagnetometer( )
php function yFirstMagnetometer( )
cpp YMagnetometer* yFirstMagnetometer( )
m YMagnetometer* yFirstMagnetometer( )
pas function yFirstMagnetometer( ): TYMagnetometer
vb function yFirstMagnetometer( ) As YMagnetometer
cs YMagnetometer FirstMagnetometer( )
java YMagnetometer FirstMagnetometer( )
py def FirstMagnetometer( )
```

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()****YMagnetometer****magnetometer→calibrateFromPoints( )**

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	- <b>(NSString*) describe</b>
pas	function <b>describe( )</b> : string
vb	function <b>describe( )</b> As String
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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)=RELAYL01-123456.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 magnetometer (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**magnetometer→get\_advertisedValue()**  
**magnetometer→advertisedValue()****magnetometer→get\_advertisedValue( )**

**YMagnetometer**

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

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

**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()**  
**magnetometer→currentRawValue()****magnetometer→get\_currentRawValue( )**

**YMagnetometer**

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

```
js function get_currentRawValue( )
nodejs function get_currentRawValue( )
php function get_currentRawValue( )
cpp double get_currentRawValue( )
m -(double) currentRawValue
pas function get_currentRawValue( ): double
vb function get_currentRawValue( ) As Double
cs double get_currentRawValue( )
java double get_currentRawValue( )
py def get_currentRawValue( )
cmd YMagnetometer target get_currentRawValue
```

**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()**  
**magnetometer→currentValue()****magnetometer→get\_currentValue( )**

**YMagnetometer**

Returns the current value of the magnetic field.

```
js function get_currentValue( )
nodejs function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YMagnetometer target get_currentValue
```

**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→getErrorMessage()**  
**magnetometer→errorMessage()magnetometer→**  
**getErrorMessage( )**

**YMagnetometer**

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

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

**js** function **get\_errorType( )****nodejs** function **get\_errorType( )****php** function **get\_errorType( )****cpp** YRETCODE **get\_errorType( )****pas** function **get\_errorType( )**: YRETCODE**vb** function **get\_errorType( )** As YRETCODE**cs** YRETCODE **get\_errorType( )****java** int **get\_errorType( )****py** def **get\_errorType( )**

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the magnetometer object

---

<b>magnetometer→get_friendlyName()</b>	<b>YMagnetometer</b>
<b>magnetometer→friendlyName()magnetometer→</b>	
<b>get_friendlyName( )</b>	

---

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

<b>js</b>	function <b>get_friendlyName( )</b>
<b>nodejs</b>	function <b>get_friendlyName( )</b>
<b>php</b>	function <b>get_friendlyName( )</b>
<b>cpp</b>	string <b>get_friendlyName( )</b>
<b>m</b>	-(NSString*) friendlyName
<b>cs</b>	string <b>get_friendlyName( )</b>
<b>java</b>	String <b>get_friendlyName( )</b>
<b>py</b>	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 exemple: 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>
node.js	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	<b>def 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()**  
**magnetometer→functionId()****magnetometer→get\_functionId( )**

**YMagnetometer**

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

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the magnetometer. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the magnetometer (ex: RELAYL01-123456.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.

<code>js</code>	function <b>get_highestValue( )</b>
<code>node.js</code>	function <b>get_highestValue( )</b>
<code>php</code>	function <b>get_highestValue( )</b>
<code>cpp</code>	double <b>get_highestValue( )</b>
<code>m</code>	-(double) <b>highestValue</b>
<code>pas</code>	function <b>get_highestValue( )</b> : double
<code>vb</code>	function <b>get_highestValue( )</b> As Double
<code>cs</code>	double <b>get_highestValue( )</b>
<code>java</code>	double <b>get_highestValue( )</b>
<code>py</code>	def <b>get_highestValue( )</b>
<code>cmd</code>	Y Magnetometer <b>target 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 get_logFrequency( )
nodejs function get_logFrequency( )
php  function get_logFrequency( )
cpp   string get_logFrequency( )
m    -(NSString*) logFrequency
pas   function get_logFrequency( ):string
vb    function get_logFrequency( ) As String
cs    string get_logFrequency( )
java  String get_logFrequency( )
py    def get_logFrequency( )
cmd   YMagnetometer target get_logFrequency
```

**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()**  
**magnetometer→logicalName()****magnetometer→get\_logicalName( )**

**YMagnetometer**

Returns the logical name of the magnetometer.

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

**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 **get\_lowestValue( )**  
nodejs    function **get\_lowestValue( )**  
php    function **get\_lowestValue( )**  
cpp    double **get\_lowestValue( )**  
m    -(double) lowestValue  
pas    function **get\_lowestValue( )**: double  
vb    function **get\_lowestValue( )** As Double  
cs    double **get\_lowestValue( )**  
java    double **get\_lowestValue( )**  
py    def **get\_lowestValue( )**  
cmd    YMagnetometer **target get\_lowestValue**

**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()**  
**magnetometer→module()magnetometer→**  
**get\_module( )**

**YMagnetometer**

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

**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 get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

```
magnetometer→get_recordedData()  
magnetometer→recordedData()magnetometer→  
get_recordedData()
```

# YMagnetometer

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

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 measurement, 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 measurement, 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()**  
**magnetometer→reportFrequency()****magnetometer→get\_reportFrequency( )**

**YMagnetometer**

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

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

**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()**  
**magnetometer→resolution()****magnetometer→get\_resolution( )**

**YMagnetometer**

Returns the resolution of the measured values.

```
js   function get_resolution( )
nodejs function get_resolution( )
php  function get_resolution( )
cpp   double get_resolution( )
m    -(double) resolution
pas   function get_resolution( ): double
vb   function get_resolution( ) As Double
cs   double get_resolution( )
java  double get_resolution( )
py   def get_resolution( )
cmd  YMagnetometer target get_resolution
```

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>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) <b>unit</b>
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	<b>Y Magnetometer 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> {
node.js	function <b>get(userData)</b> {
php	function <b>get(userData)</b> {
cpp	void * <b>get(userData)</b> {
m	-(void*) <b>get(userData)</b> {
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()**  
**magnetometer→xValue()magnetometer→**  
**get\_xValue( )**

**YMagnetometer**

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

**js** function **get\_xValue( )**  
**nodejs** function **get\_xValue( )**  
**php** function **get\_xValue( )**  
**cpp** double **get\_xValue( )**  
**m** -(double) **xValue**  
**pas** function **get\_xValue( )**: double  
**vb** function **get\_xValue( )** As Double  
**cs** double **get\_xValue( )**  
**java** double **get\_xValue( )**  
**py** def **get\_xValue( )**  
**cmd** YMagnetometer **target get\_xValue**

**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 get_yValue( )  
nodejs function get_yValue( )  
php  function get_yValue( )  
cpp   double get_yValue( )  
m    -(double) yValue  
pas   function get_yValue( ): double  
vb    function get_yValue( ) As Double  
cs    double get_yValue( )  
java  double get_yValue( )  
py    def get_yValue( )  
cmd   YMagnetometer target get_yValue
```

**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	<b>Y Magnetometer target 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 isOnline( )
nodejs function isOnline( )
php  function isOnline( )
cpp   bool isOnline( )
m    -(BOOL) isOnline
pas   function isOnline( ): boolean
vb    function isOnline( ) As Boolean
cs   bool isOnline( )
java  boolean isOnline( )
py    def isOnline( )
```

If there is a cached value for the 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()****YMagnetometer**

Preloads the magnetometer cache with a specified validity duration.

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

**magnetometer→loadCalibrationPoints()****YMagnetometer****magnetometer→loadCalibrationPoints( )**

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  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( )**

**Y Magnetometer**

Continues the enumeration of magnetometers started using **yFirstMagnetometer( )**.

<b>js</b>	function <b>nextMagnetometer( )</b>
<b>node.js</b>	function <b>nextMagnetometer( )</b>
<b>php</b>	function <b>nextMagnetometer( )</b>
<b>cpp</b>	<b>Y Magnetometer * nextMagnetometer( )</b>
<b>m</b>	-( <b>Y Magnetometer*</b> ) <b>nextMagnetometer</b>
<b>pas</b>	function <b>nextMagnetometer( )</b> : <b>TY Magnetometer</b>
<b>vb</b>	function <b>nextMagnetometer( )</b> As <b>Y Magnetometer</b>
<b>cs</b>	<b>Y Magnetometer</b> <b>nextMagnetometer( )</b>
<b>java</b>	<b>Y Magnetometer</b> <b>nextMagnetometer( )</b>
<b>py</b>	def <b>nextMagnetometer( )</b>

**Returns :**

a pointer to a **Y Magnetometer** object, corresponding to a magnetometer currently online, or a null pointer if there are no more magnetometers to enumerate.

**magnetometer→registerTimedReportCallback()****YMagnetometer****magnetometer→****registerTimedReportCallback( )**

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

<b>js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>node.js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>php</b>	function registerTimedReportCallback( <b>\$callback</b> )
<b>cpp</b>	int registerTimedReportCallback( YMagnetometerTimedReportCallback <b>callback</b> )
<b>m</b>	-(int) registerTimedReportCallback : (YMagnetometerTimedReportCallback) <b>callback</b>
<b>pas</b>	function registerTimedReportCallback( <b>callback</b> : TYMagnetometerTimedReportCallback): LongInt
<b>vb</b>	function registerTimedReportCallback( ) As Integer
<b>cs</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>java</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>py</b>	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YMagnetometerValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YMagnetometerValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYMagnetometerValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()** YMagnetometer  
**magnetometer→setHighestValue()** **magnetometer→set\_highestValue()**

Changes the recorded maximal value observed.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YMagnetometer target set_highestValue newval
```

**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()** **YMagnetometer**  
**magnetometer→setLogFrequency()magnetometer→**  
**set\_logFrequency( )**

Changes the datalogger recording frequency for this function.

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

---

<b>magnetometer→set_logicalName()</b> <b>magnetometer→setLogicalName()</b> <b>magnetometer→</b> <b>set_logicalName( )</b>	<b>YMagnetometer</b>
---	----------------------

---

Changes the logical name of the magnetometer.

js	function <b>set_logicalName( newval)</b>
nodejs	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	<b>YMagnetometer 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()**  
**magnetometer→setLowestValue()****magnetometer→set\_lowestValue()**

**YMagnetometer**

Changes the recorded minimal value observed.

```
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 YMagnetometer target set_lowestValue newval
```

**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.

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

**magnetometer→set\_resolution()**  
**magnetometer→setResolution()****magnetometer→set\_resolution()**

**YMagnetometer**

Changes the resolution of the measured physical values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YMagnetometer target set_resolution newval
```

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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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>
node.js	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()**  
**measure→averageValue()** measure→  
**get\_averageValue( )**

**YMeasure**

Returns the average value observed during the time interval covered by this measure.

js    function **get\_averageValue( )**  
nodejs    function **get\_averageValue( )**  
php    function **get\_averageValue( )**  
cpp    double **get\_averageValue( )**  
m    -(double) **averageValue**  
pas    function **get\_averageValue( )**: double  
vb    function **get\_averageValue( )** As Double  
cs    double **get\_averageValue( )**  
java    double **get\_averageValue( )**  
py    def **get\_averageValue( )**

**Returns :**

a floating-point number corresponding to the average value observed.

**measure→get\_endTimeUTC()**  
**measure→endTimeUTC()measure→**  
**get\_endTimeUTC( )**

**YMeasure**

Returns the end time of the measure, relative to the Jan 1, 1970 UTC (Unix timestamp).

js	function <b>get_endTimeUTC( )</b>
node.js	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	<b>def get_endTimeUTC( )</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 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>
node.js	function <b>get_maxValue( )</b>
php	function <b>get_maxValue( )</b>
cpp	double <b>get_maxValue( )</b>
m	-(double) maxValue
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( ): double</b>
vb	function <b>get_minValue( ) As Double</b>
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→getStartTimeUTC()**  
**measure→startTimeUTC()** measure→  
**getStartTimeUTC( )**

**YMeasure**

Returns the start time of the measure, relative to the Jan 1, 1970 UTC (Unix timestamp).

js    function **getStartTimeUTC( )**  
nodejs    function **getStartTimeUTC( )**  
php    function **getStartTimeUTC( )**  
cpp    double **getStartTimeUTC( )**  
m    -(double) **startTimeUTC**  
pas    function **getStartTimeUTC( )**: double  
vb    function **getStartTimeUTC( )** As Double  
cs    double **getStartTimeUTC( )**  
java    double **getStartTimeUTC( )**  
py    def **getStartTimeUTC( )**

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>
node.js	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

#### 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→get_lastLogs()</b> Returns a string with last logs of the module.
<b>module→get_logicalName()</b> Returns the logical name of the module.
<b>module→get_luminosity()</b> Returns the luminosity of the module informative leds (from 0 to 100).
<b>module→get_persistentSettings()</b> Returns the current state of persistent module settings.
<b>module→get_productId()</b> Returns the USB device identifier of the module.
<b>module→get_productName()</b> Returns the commercial name of the module, as set by the factory.
<b>module→get_productRelease()</b> Returns the hardware release version of the module.
<b>module→get_rebootCountdown()</b> Returns the remaining number of seconds before the module restarts, or zero when no reboot has been scheduled.
<b>module→get_serialNumber()</b> Returns the serial number of the module, as set by the factory.
<b>module→get_upTime()</b> Returns the number of milliseconds spent since the module was powered on.
<b>module→get_usbBandwidth()</b> Returns the number of USB interfaces used by the module.
<b>module→get_usbCurrent()</b> Returns the current consumed by the module on the USB bus, in milli-amps.
<b>module→get(userData)</b> Returns the value of the userData attribute, as previously stored using method <code>set(userData)</code> .
<b>module→isOnline()</b> Checks if the module is currently reachable, without raising any error.
<b>module→isOnline_async(callback, context)</b> Checks if the module is currently reachable, without raising any error.
<b>module→load(msValidity)</b> Preloads the module cache with a specified validity duration.
<b>module→load_async(msValidity, callback, context)</b> Preloads the module cache with a specified validity duration (asynchronous version).
<b>module→nextModule()</b> Continues the module enumeration started using <code>yFirstModule()</code> .
<b>module→reboot(secBeforeReboot)</b> Schedules a simple module reboot after the given number of seconds.
<b>module→registerLogCallback(callback)</b> todo
<b>module→revertFromFlash()</b> Reloads the settings stored in the nonvolatile memory, as when the module is powered on.
<b>module→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()yFindModule( )

**YModule**

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

js	function <b>yFindModule( func)</b>
node.js	function <b>FindModule( func)</b>
php	function <b>yFindModule( \$func)</b>
cpp	YModule* <b>yFindModule( string func)</b>
m	+ <b>(YModule*) yFindModule : (NSString*) func</b>
pas	function <b>yFindModule( func: string): TYModule</b>
vb	function <b>yFindModule( ByVal func As String) As YModule</b>
cs	YModule <b>FindModule( string func)</b>
java	YModule <b>FindModule( String func)</b>
py	def <b>FindModule( 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()yFirstModule( )**

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( ): TYModule</b>
vb	function <b>yFirstModule( ) As YModule</b>
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()****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	- <b>(NSString*) describe</b>
pas	function <b>describe( )</b> : string
vb	function <b>describe( )</b> As String
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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()****YModule**

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

js	function <b>download( pathname)</b>
node.js	function <b>download( pathname)</b>
php	function <b>download( \$pathname)</b>
cpp	string <b>download( string pathname)</b>
m	- <b>(NSData*) download : (NSString*) pathname</b>
pas	function <b>download( pathname: string): TByteArray</b>
vb	function <b>download( ) As Byte</b>
py	<b>def download( pathname)</b>
cmd	<b>YModule target download 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 <b>functionCount( )</b>
node.js	function <b>functionCount( )</b>
php	function <b>functionCount( )</b>
cpp	int <b>functionCount( )</b>
m	- <b>(int) functionCount</b>
pas	function <b>functionCount( )</b> : integer
vb	function <b>functionCount( )</b> As Integer
cs	int <b>functionCount( )</b>
py	def <b>functionCount( )</b>

**Returns :**

the number of functions on the module

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

**module→functionId()****YModule**

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

js	function <b>functionId(</b> <b>functionIndex)</b>
node.js	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	- <b>(NSString*) functionId : (int) functionIndex</b>
pas	function <b>functionId(</b> <b>functionIndex:</b> integer): string
vb	function <b>functionId(</b> <b>ByVal functionIndex As Integer)</b> 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()****YModule**

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

```
js  function functionName( functionIndex)
nodejs function functionName( functionIndex)
php  function functionName( $functionIndex)
cpp   string functionName( int functionIndex)
m    -(NSString*) functionName : (int) functionIndex
pas   function functionName( functionIndex: integer): string
vb    function functionName( ByVal functionIndex As Integer) As String
cs    string functionName( int functionIndex)
py    def functionName( functionIndex)
```

**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( functionIndex)</b>
nodejs	function <b>functionValue( functionIndex)</b>
php	function <b>functionValue( \$functionIndex)</b>
cpp	string <b>functionValue( int functionIndex)</b>
m	<b>-(NSString*) functionValue : (int) functionIndex</b>
pas	function <b>functionValue( functionIndex: integer): string</b>
vb	function <b>functionValue( ByVal functionIndex As Integer) As String</b>
cs	string <b>functionValue( int functionIndex)</b>
py	<b>def functionValue( 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()**  
**module→beacon()module→get\_beacon( )**

**YModule**

Returns the state of the localization beacon.

js    function **get\_beacon( )**  
node.js    function **get\_beacon( )**  
php    function **get\_beacon( )**  
cpp    Y\_BEACON\_enum **get\_beacon( )**  
m    -(Y\_BEACON\_enum) **beacon**  
pas    function **get\_beacon( ): Integer**  
vb    function **get\_beacon( ) As Integer**  
cs    int **get\_beacon( )**  
java    int **get\_beacon( )**  
py    def **get\_beacon( )**  
cmd    YModule **target get\_beacon**

**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→getErrorMessage()**  
**module→errorMessage()module→**  
**getErrorMessage( )**

**YModule**

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

<b>js</b>	function <b>getErrorMessage( )</b>
<b>nodejs</b>	function <b>getErrorMessage( )</b>
<b>php</b>	function <b>getErrorMessage( )</b>
<b>cpp</b>	string <b>getErrorMessage( )</b>
<b>m</b>	-(NSString*) errorMessage
<b>pas</b>	function <b>getErrorMessage( )</b> : string
<b>vb</b>	function <b>getErrorMessage( )</b> As String
<b>cs</b>	string <b>getErrorMessage( )</b>
<b>java</b>	String <b>getErrorMessage( )</b>
<b>py</b>	def <b>getErrorMessage( )</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()**  
**module→errorType()module→get\_errorType( )****YModule**

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

js	function <b>get_errorType( )</b>
node.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 module object

**module→get\_firmwareRelease()**  
**module→firmwareRelease()****module→get\_firmwareRelease( )**

**YModule**

Returns the version of the firmware embedded in the module.

<b>js</b>	function <b>get_firmwareRelease( )</b>
<b>nodejs</b>	function <b>get_firmwareRelease( )</b>
<b>php</b>	function <b>get_firmwareRelease( )</b>
<b>cpp</b>	string <b>get_firmwareRelease( )</b>
<b>m</b>	-(NSString*) firmwareRelease
<b>pas</b>	function <b>get_firmwareRelease( )</b> : string
<b>vb</b>	function <b>get_firmwareRelease( )</b> As String
<b>cs</b>	string <b>get_firmwareRelease( )</b>
<b>java</b>	String <b>get_firmwareRelease( )</b>
<b>py</b>	def <b>get_firmwareRelease( )</b>
<b>cmd</b>	<b>YModule target 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>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	- <b>(NSString*) hardwareId</b>
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	<b>String get_hardwareId( )</b>
py	<b>def 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()**  
**module→icon2d()module→get\_icon2d( )****YModule**

Returns the icon of the module.

js	function <b>get_icon2d( )</b>
node.js	function <b>get_icon2d( )</b>
php	function <b>get_icon2d( )</b>
cpp	string <b>get_icon2d( )</b>
m	-(NSData*) <b>icon2d</b>
pas	function <b>get_icon2d( ): TByteArray</b>
vb	function <b>get_icon2d( ) As Byte</b>
py	<b>def get_icon2d( )</b>
cmd	<b>YModule target 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()**  
**module→lastLogs()****module→get\_lastLogs( )****YModule**

Returns a string with last logs of the module.

js	function <b>get_lastLogs( )</b>
node.js	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( ): string</b>
vb	function <b>get_lastLogs( ) As String</b>
cs	string <b>get_lastLogs( )</b>
java	String <b>get_lastLogs( )</b>
py	def <b>get_lastLogs( )</b>
cmd	YModule target <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()**  
**module→logicalName()****module→get\_logicalName( )**

**YModule**

Returns the logical name of the module.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	<b>YModule target 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>
node.js	function <b>get_luminosity()</b>
php	function <b>get_luminosity()</b>
cpp	int <b>get_luminosity()</b>
m	-(int) <b>luminosity</b>
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	<b>YModule target 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()**  
**module→persistentSettings()module→**  
**get\_persistentSettings( )**

**YModule**

Returns the current state of persistent module settings.

<b>js</b>	function <b>get_persistentSettings( )</b>
<b>nodejs</b>	function <b>get_persistentSettings( )</b>
<b>php</b>	function <b>get_persistentSettings( )</b>
<b>cpp</b>	Y_PERSISTENTSETTINGS_enum <b>get_persistentSettings( )</b>
<b>m</b>	-(Y_PERSISTENTSETTINGS_enum) persistentSettings
<b>pas</b>	function <b>get_persistentSettings( )</b> : Integer
<b>vb</b>	function <b>get_persistentSettings( )</b> As Integer
<b>cs</b>	int <b>get_persistentSettings( )</b>
<b>java</b>	int <b>get_persistentSettings( )</b>
<b>py</b>	def <b>get_persistentSettings( )</b>
<b>cmd</b>	<b>YModule target 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>
node.js	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 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()**  
**module→productName()** module→  
**get\_productName( )**

**YModule**

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

<b>js</b>	function <b>get_productName( )</b>
<b>nodejs</b>	function <b>get_productName( )</b>
<b>php</b>	function <b>get_productName( )</b>
<b>cpp</b>	string <b>get_productName( )</b>
<b>m</b>	-(NSString*) productName
<b>pas</b>	function <b>get_productName( )</b> : string
<b>vb</b>	function <b>get_productName( )</b> As String
<b>cs</b>	string <b>get_productName( )</b>
<b>java</b>	String <b>get_productName( )</b>
<b>py</b>	def <b>get_productName( )</b>
<b>cmd</b>	<b>YModule target 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()**  
**module→productRelease()** module→  
**get\_productRelease( )**

**YModule**

Returns the hardware release version of the module.

<b>js</b>	function <b>get_productRelease( )</b>
<b>nodejs</b>	function <b>get_productRelease( )</b>
<b>php</b>	function <b>get_productRelease( )</b>
<b>cpp</b>	int <b>get_productRelease( )</b>
<b>m</b>	-(int) productRelease
<b>pas</b>	function <b>get_productRelease( )</b> : LongInt
<b>vb</b>	function <b>get_productRelease( )</b> As Integer
<b>cs</b>	int <b>get_productRelease( )</b>
<b>java</b>	int <b>get_productRelease( )</b>
<b>py</b>	def <b>get_productRelease( )</b>
<b>cmd</b>	<b>YModule target 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()**  
**module→rebootCountdown()****module→get\_rebootCountdown( )**

**YModule**

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

<b>js</b>	function <b>get_rebootCountdown( )</b>
<b>nodejs</b>	function <b>get_rebootCountdown( )</b>
<b>php</b>	function <b>get_rebootCountdown( )</b>
<b>cpp</b>	int <b>get_rebootCountdown( )</b>
<b>m</b>	-(int) <b>rebootCountdown</b>
<b>pas</b>	function <b>get_rebootCountdown( ): LongInt</b>
<b>vb</b>	function <b>get_rebootCountdown( ) As Integer</b>
<b>cs</b>	int <b>get_rebootCountdown( )</b>
<b>java</b>	int <b>get_rebootCountdown( )</b>
<b>py</b>	def <b>get_rebootCountdown( )</b>
<b>cmd</b>	<b>YModule target 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()**  
**module→serialNumber()** **module→get\_serialNumber( )**

**YModule**

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*) <b>serialNumber</b>
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 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( ): int64</b>
vb	function <b>get_upTime( ) As Long</b>
cs	long <b>get_upTime( )</b>
java	long <b>get_upTime( )</b>
py	def <b>get_upTime( )</b>
cmd	<b>YModule target 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()**  
**module→usbBandwidth()** module→  
**get\_usbBandwidth( )**

**YModule**

Returns the number of USB interfaces used by the module.

```
js function get_usbBandwidth( )
nodejs function get_usbBandwidth( )
php function get_usbBandwidth( )
cpp Y_USBWIDTH_enum get_usbBandwidth( )
m -(Y_USBWIDTH_enum) usbBandwidth
pas function get_usbBandwidth( ): Integer
vb function get_usbBandwidth( ) As Integer
cs int get_usbBandwidth( )
java int get_usbBandwidth( )
py def get_usbBandwidth( )
cmd YModule target get_usbBandwidth
```

**Returns :**

either Y\_USBWIDTH\_SIMPLE or Y\_USBWIDTH\_DOUBLE, according to the number of USB interfaces used by the module

On failure, throws an exception or returns Y\_USBWIDTH\_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) <b>usbCurrent</b>
pas	function <b>get_usbCurrent( ): LongInt</b>
vb	function <b>get_usbCurrent( ) As Integer</b>
cs	int <b>get_usbCurrent( )</b>
java	int <b>get_usbCurrent( )</b>
py	def <b>get_usbCurrent( )</b>
cmd	<b>YModule target 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→get(userData)()**

Returns the value of the userData attribute, as previously stored using method set(userData).

js	function <b>get(userData)</b> ( )
node.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.

**module→isOnline()****YModule**

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

js	function <b>isOnline( )</b>
node.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 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()****YModule**

Preloads the module cache with a specified validity duration.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</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 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()****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	- <b>(YModule*) 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()****YModule**

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

```
js function reboot( secBeforeReboot)
nodejs function reboot( secBeforeReboot)
php function reboot( $secBeforeReboot)
cpp int reboot( int secBeforeReboot)
m -(int) reboot : (int) secBeforeReboot
pas function reboot( secBeforeReboot: LongInt): LongInt
vb function reboot( ) As Integer
cs int reboot( int secBeforeReboot)
java int reboot( int secBeforeReboot)
py def reboot( secBeforeReboot)
cmd YModule target reboot secBeforeReboot
```

**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 registerLogCallback( YModuleLogCallback <b>callback</b> )
m	-(void) registerLogCallback : (YModuleLogCallback) <b>callback</b>
vb	function registerLogCallback( ByVal <b>callback</b> As YModuleLogCallback) As Integer
cs	int registerLogCallback( LogCallback <b>callback</b> )
py	def registerLogCallback( <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 revertFromFlash( )
node.js	function revertFromFlash( )
php	function revertFromFlash( )
cpp	int revertFromFlash( )
m	-(int) revertFromFlash
pas	function revertFromFlash( ): LongInt
vb	function revertFromFlash( ) As Integer
cs	int revertFromFlash( )
java	int revertFromFlash( )
py	def revertFromFlash( )
cmd	YModule target revertFromFlash

**Returns :**

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

**module→saveToFlash()****YModule**

Saves current settings in the nonvolatile memory of the module.

js	function <b>saveToFlash( )</b>
node.js	function <b>saveToFlash( )</b>
php	function <b>saveToFlash( )</b>
cpp	int <b>saveToFlash( )</b>
m	- <b>(int) 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 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()**  
**module→setBeacon()****module→set\_beacon( )****YModule**

Turns on or off the module localization beacon.

js	function <b>set_beacon( newval)</b>
node.js	function <b>set_beacon( newval)</b>
php	function <b>set_beacon( \$newval)</b>
cpp	int <b>set_beacon( Y_BEACON_enum newval)</b>
m	-(int) setBeacon : (Y_BEACON_enum) <b>newval</b>
pas	function <b>set_beacon( newval: Integer): integer</b>
vb	function <b>set_beacon( ByVal newval As Integer) As Integer</b>
cs	int <b>set_beacon( int newval)</b>
java	int <b>set_beacon( int newval)</b>
py	def <b>set_beacon( newval)</b>
cmd	<b>YModule 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()**  
**module->setLogicalName()****module->**  
**set\_logicalName( )**

**YModule**


---

Changes the logical name of the module.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YModule 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 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( newval)</b>
nodejs	function <b>set_luminosity( newval)</b>
php	function <b>set_luminosity( \$newval)</b>
cpp	int <b>set_luminosity( int newval)</b>
m	-(int) setLuminosity : (int) <b>newval</b>
pas	function <b>set_luminosity( newval: LongInt): integer</b>
vb	function <b>set_luminosity( ByVal newval As Integer) As Integer</b>
cs	int <b>set_luminosity( int newval)</b>
java	int <b>set_luminosity( int newval)</b>
py	def <b>set_luminosity( 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.

<b>js</b>	function <b>set_usbBandwidth( newval)</b>
<b>nodejs</b>	function <b>set_usbBandwidth( newval)</b>
<b>php</b>	function <b>set_usbBandwidth( \$newval)</b>
<b>cpp</b>	int <b>set_usbBandwidth( Y_USBBANDWIDTH_enum newval)</b>
<b>m</b>	-(int) <b>setUsbBandwidth : (Y_USBBANDWIDTH_enum) newval</b>
<b>pas</b>	function <b>set_usbBandwidth( newval: Integer): integer</b>
<b>vb</b>	function <b>set_usbBandwidth( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_usbBandwidth( int newval)</b>
<b>java</b>	int <b>set_usbBandwidth( int newval)</b>
<b>py</b>	def <b>set_usbBandwidth( newval)</b>
<b>cmd</b>	<b>YModule target set_usbBandwidth 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)**  
**module→setUserData()****module→set(userData( ))****YModule**

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

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

---

<b>module</b> → <b>triggerFirmwareUpdate()</b> <b>module</b> → <b>triggerFirmwareUpdate( )</b>	<b>YModule</b>
---	----------------

---

Schedules a module reboot into special firmware update mode.

```
js function triggerFirmwareUpdate( secBeforeReboot)
nodejs function triggerFirmwareUpdate( secBeforeReboot)
php function triggerFirmwareUpdate( $secBeforeReboot)
cpp int triggerFirmwareUpdate( int secBeforeReboot)
m -(int) triggerFirmwareUpdate : (int) secBeforeReboot
pas function triggerFirmwareUpdate( secBeforeReboot: LongInt): LongInt
vb function triggerFirmwareUpdate( ) As Integer
cs int triggerFirmwareUpdate( int secBeforeReboot)
java int triggerFirmwareUpdate( int secBeforeReboot)
py def triggerFirmwareUpdate( secBeforeReboot)
cmd YModule target triggerFirmwareUpdate secBeforeReboot
```

**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>
node.js	var yoctolib = require('yoctolib');
	var YNetwork = yoctolib.YNetwork;
php	require_once('yocto_network.php');
cpp	#include "yocto_network.h"
m	#import "yocto_network.h"
pas	uses yocto_network;
vb	yocto_network.vb
cs	yocto_network.cs
java	import com.yoctopuce.YoctoAPI.YNetwork;
py	from yocto_network import *

### Global functions

#### yFindNetwork(func)

Retrieves a network interface for a given identifier.

#### yFirstNetwork()

Starts the enumeration of network interfaces currently accessible.

### YNetwork methods

#### network→callbackLogin(username, password)

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

#### network→describe()

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

#### network→get\_adminPassword()

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

#### network→get\_advertisedValue()

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

#### network→get\_callbackCredentials()

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

#### network→get\_callbackEncoding()

Returns the encoding standard to use for representing notification values.

#### network→get\_callbackMaxDelay()

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

#### network→get\_callbackMethod()

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

#### network→get\_callbackMinDelay()

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

#### network→get\_callbackUrl()

Returns the callback URL to notify of significant state changes.

#### network→get\_discoverable()

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

### 3. Reference

<b>network→get_errorMessage()</b>	Returns the error message of the latest error with the network interface.
<b>network→get_errorType()</b>	Returns the numerical error code of the latest error with the network interface.
<b>network→get_friendlyName()</b>	Returns a global identifier of the network interface in the format MODULE_NAME . FUNCTION_NAME.
<b>network→get_functionDescriptor()</b>	Returns a unique identifier of type YFUN_DESCR corresponding to the function.
<b>network→get_functionId()</b>	Returns the hardware identifier of the network interface, without reference to the module.
<b>network→get_hardwareId()</b>	Returns the unique hardware identifier of the network interface in the form SERIAL . FUNCTIONID.
<b>network→get_ipAddress()</b>	Returns the IP address currently in use by the device.
<b>network→get_logicalName()</b>	Returns the logical name of the network interface.
<b>network→get_macAddress()</b>	Returns the MAC address of the network interface.
<b>network→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>network→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>network→get_poeCurrent()</b>	Returns the current consumed by the module from Power-over-Ethernet (PoE), in milli-amps.
<b>network→get_primaryDNS()</b>	Returns the IP address of the primary name server to be used by the module.
<b>network→get_readiness()</b>	Returns the current established working mode of the network interface.
<b>network→get_router()</b>	Returns the IP address of the router on the device subnet (default gateway).
<b>network→get_secondaryDNS()</b>	Returns the IP address of the secondary name server to be used by the module.
<b>network→get_subnetMask()</b>	Returns the subnet mask currently used by the device.
<b>network→get_userData()</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>network→get_userPassword()</b>	Returns a hash string if a password has been set for "user" user, or an empty string otherwise.
<b>network→get_wwwWatchdogDelay()</b>	Returns the allowed downtime of the WWW link (in seconds) before triggering an automated reboot to try to recover Internet connectivity.
<b>network→isOnline()</b>	Checks if the network interface is currently reachable, without raising any error.
<b>network→isOnline_async(callback, context)</b>	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()yFindNetwork( )**

Retrieves a network interface for a given identifier.

<code>js</code>	<code>function yFindNetwork( func)</code>
<code>node.js</code>	<code>function FindNetwork( func)</code>
<code>php</code>	<code>function yFindNetwork( \$func)</code>
<code>cpp</code>	<code>YNetwork* yFindNetwork( const string&amp; func)</code>
<code>m</code>	<code>YNetwork* yFindNetwork( NSString* func)</code>
<code>pas</code>	<code>function yFindNetwork( func: string): TYNetwork</code>
<code>vb</code>	<code>function yFindNetwork( ByVal func As String) As YNetwork</code>
<code>cs</code>	<code>YNetwork FindNetwork( string func)</code>
<code>java</code>	<code>YNetwork FindNetwork( String func)</code>
<code>py</code>	<code>def FindNetwork( 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 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()yFirstNetwork( )**

Starts the enumeration of network interfaces currently accessible.

js	function <b>yFirstNetwork( )</b>
node.js	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( username, password)</b>
nodejs	function <b>callbackLogin( username, password)</b>
php	function <b>callbackLogin( \$username, \$password)</b>
cpp	int <b>callbackLogin( string username, string password)</b>
m	- <b>(int) callbackLogin : (NSString*) username : (NSString*) password</b>
pas	function <b>callbackLogin( username: string, password: string): integer</b>
vb	function <b>callbackLogin( ByVal username As String,</b> <b>                  ByVal password As String) As Integer</b>
cs	int <b>callbackLogin( string username, string password)</b>
java	int <b>callbackLogin( String username, String password)</b>
py	def <b>callbackLogin( username, password)</b>
cmd	YNetwork target <b>callbackLogin</b> <b>username</b> <b>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()****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 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**network→get\_adminPassword()**  
**network→adminPassword()network→**  
**get\_adminPassword( )**

**YNetwork**

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

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

**YNetwork**

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

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

**Returns :**

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

**network→get\_callbackCredentials()****YNetwork****network→callbackCredentials()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>
node.js	function <b>get_callbackCredentials( )</b>
php	function <b>get_callbackCredentials( )</b>
cpp	string <b>get_callbackCredentials( )</b>
m	-(NSString*) <b>callbackCredentials</b>
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 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()**  
**network→callbackEncoding()network→**  
**get\_callbackEncoding( )**

**YNetwork**

Returns the encoding standard to use for representing notification values.

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

**Returns :**

a value among Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV and Y\_CALLBACKENCODING\_YOCTO\_API corresponding to the encoding standard to use for representing notification values

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

**network→get\_callbackMaxDelay()**  
**network→callbackMaxDelay()network→**  
**get\_callbackMaxDelay( )**

**YNetwork**

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

<b>js</b>	function <b>get_callbackMaxDelay( )</b>
<b>node.js</b>	function <b>get_callbackMaxDelay( )</b>
<b>php</b>	function <b>get_callbackMaxDelay( )</b>
<b>cpp</b>	int <b>get_callbackMaxDelay( )</b>
<b>m</b>	-(int) <b>callbackMaxDelay</b>
<b>pas</b>	function <b>get_callbackMaxDelay( )</b> : LongInt
<b>vb</b>	function <b>get_callbackMaxDelay( )</b> As Integer
<b>cs</b>	int <b>get_callbackMaxDelay( )</b>
<b>java</b>	int <b>get_callbackMaxDelay( )</b>
<b>py</b>	def <b>get_callbackMaxDelay( )</b>
<b>cmd</b>	<b>YNetwork target 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()**  
**network→callbackMethod()network→**  
**get\_callbackMethod( )**

**YNetwork**

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

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

**Returns :**

a value among Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET and Y\_CALLBACKMETHOD\_PUT corresponding to the HTTP method used to notify callbacks for significant state changes

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

**network→get\_callbackMinDelay()**  
**network→callbackMinDelay()network→**  
**get\_callbackMinDelay( )**

**YNetwork**

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

<b>js</b>	function <b>get_callbackMinDelay( )</b>
<b>nodejs</b>	function <b>get_callbackMinDelay( )</b>
<b>php</b>	function <b>get_callbackMinDelay( )</b>
<b>cpp</b>	int <b>get_callbackMinDelay( )</b>
<b>m</b>	-(int) <b>callbackMinDelay</b>
<b>pas</b>	function <b>get_callbackMinDelay( )</b> : LongInt
<b>vb</b>	function <b>get_callbackMinDelay( )</b> As Integer
<b>cs</b>	int <b>get_callbackMinDelay( )</b>
<b>java</b>	int <b>get_callbackMinDelay( )</b>
<b>py</b>	def <b>get_callbackMinDelay( )</b>
<b>cmd</b>	<b>YNetwork target 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()**  
**network→callbackUrl()network→**  
**get\_callbackUrl( )**

**YNetwork**

Returns the callback URL to notify of significant state changes.

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

**Returns :**

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

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

**network→get\_discoverable()**  
**network→discoverable()network→**  
**get\_discoverable()**

**YNetwork**

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

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

**Returns :**

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

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

**network→getErrorMessage()**  
**network→errorMessage()network→getErrorMessage( )**

**YNetwork**

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

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

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

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

**Returns :**

a string that uniquely identifies the network interface using logical names (ex: MyCustomName.relay1) On failure, throws an exception or returns Y\_FRIENDLYNAME\_INVALID.

**network→get\_functionDescriptor()**  
**network→functionDescriptor()network→**  
**get\_functionDescriptor( )**

**YNetwork**

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

js	function <b>get_functionDescriptor()</b>
node.js	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.

**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>
node.js	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()**  
**network→hardwareId()network→**  
**get\_hardwareId( )**

**YNetwork**

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

js	function <b>get_hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the network interface (ex: RELAYL01-123456.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>
node.js	function <b>get_ipAddress( )</b>
php	function <b>get_ipAddress( )</b>
cpp	string <b>get_ipAddress( )</b>
m	- <b>(NSString*) ipAddress</b>
pas	function <b>get_ipAddress( )</b> : string
vb	function <b>get_ipAddress( )</b> As String
cs	string <b>get_ipAddress( )</b>
java	<b>String get_ipAddress( )</b>
py	def <b>get_ipAddress( )</b>
cmd	<b>YNetwork target 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()**  
**network→logicalName()network→**  
**get\_logicalName( )**

**YNetwork**

Returns the logical name of the network interface.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	<b>YNetwork target 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()**  
**network→macAddress()** network→  
**get\_macAddress( )**

**YNetwork**

Returns the MAC address of the network interface.

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

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

**Returns :**

a string corresponding to the MAC address of the network interface

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

**network→get\_module()****YNetwork****network→module()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 online.

**Returns :**

an instance of YModule

**network→get\_module\_async()**  
**network→module\_async()****YNetwork**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**network→get\_poeCurrent()**  
**network→poeCurrent()network→**  
**get\_poeCurrent( )**

**YNetwork**

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

<b>js</b>	function <b>get_poeCurrent( )</b>
<b>node.js</b>	function <b>get_poeCurrent( )</b>
<b>php</b>	function <b>get_poeCurrent( )</b>
<b>cpp</b>	int <b>get_poeCurrent( )</b>
<b>m</b>	-(int) <b>poeCurrent</b>
<b>pas</b>	function <b>get_poeCurrent( )</b> : LongInt
<b>vb</b>	function <b>get_poeCurrent( )</b> As Integer
<b>cs</b>	int <b>get_poeCurrent( )</b>
<b>java</b>	int <b>get_poeCurrent( )</b>
<b>py</b>	def <b>get_poeCurrent( )</b>
<b>cmd</b>	<b>YNetwork target 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 get_primaryDNS( )
nodejs	function get_primaryDNS( )
php	function get_primaryDNS( )
cpp	string get_primaryDNS( )
m	-(NSString*) primaryDNS
pas	function get_primaryDNS( ): string
vb	function get_primaryDNS( ) As String
cs	string get_primaryDNS( )
java	String get_primaryDNS( )
py	def get_primaryDNS( )
cmd	YNetwork target get_primaryDNS

**Returns :**

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

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

**network→get\_readiness()****YNetwork****network→readiness()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	<b>Y_READINESS_enum get_readiness( )</b>
m	-( <b>Y_READINESS_enum</b> ) <b>readiness</b>
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	<b>YNetwork target 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()**  
**network→router()network→get\_router( )****YNetwork**

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

js	function <b>get_router( )</b>
node.js	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( ): string</b>
vb	function <b>get_router( ) As String</b>
cs	string <b>get_router( )</b>
java	<b>String get_router( )</b>
py	def <b>get_router( )</b>
cmd	<b>YNetwork target 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()**  
**network→secondaryDNS()network→**  
**get\_secondaryDNS( )**

**YNetwork**

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

<b>js</b>	function <b>get_secondaryDNS( )</b>
<b>nodejs</b>	function <b>get_secondaryDNS( )</b>
<b>php</b>	function <b>get_secondaryDNS( )</b>
<b>cpp</b>	string <b>get_secondaryDNS( )</b>
<b>m</b>	-(NSString*) secondaryDNS
<b>pas</b>	function <b>get_secondaryDNS( )</b> : string
<b>vb</b>	function <b>get_secondaryDNS( )</b> As String
<b>cs</b>	string <b>get_secondaryDNS( )</b>
<b>java</b>	String <b>get_secondaryDNS( )</b>
<b>py</b>	def <b>get_secondaryDNS( )</b>
<b>cmd</b>	<b>YNetwork target 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()**  
**network→subnetMask()** network→  
**get\_subnetMask( )**

**YNetwork**

Returns the subnet mask currently used by the device.

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

**Returns :**

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

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

**network→get(userData)****YNetwork****network→userData()network→get(userData())**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

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

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( )**

**YNetwork**

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

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

**Returns :**

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

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

<b>network→get_wwwWatchdogDelay()</b>	<b>YNetwork</b>
<b>network→wwwWatchdogDelay()network→</b>	
<b>get_wwwWatchdogDelay( )</b>	

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

<b>js</b>	function <b>get_wwwWatchdogDelay( )</b>
<b>nodejs</b>	function <b>get_wwwWatchdogDelay( )</b>
<b>php</b>	function <b>get_wwwWatchdogDelay( )</b>
<b>cpp</b>	int <b>get_wwwWatchdogDelay( )</b>
<b>m</b>	-(int) wwwWatchdogDelay
<b>pas</b>	function <b>get_wwwWatchdogDelay( ): LongInt</b>
<b>vb</b>	function <b>get_wwwWatchdogDelay( ) As Integer</b>
<b>cs</b>	int <b>get_wwwWatchdogDelay( )</b>
<b>java</b>	int <b>get_wwwWatchdogDelay( )</b>
<b>py</b>	def <b>get_wwwWatchdogDelay( )</b>
<b>cmd</b>	<b>YNetwork target 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	- <b>(BOOL) 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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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	<b>YNetwork * nextNetwork()</b>
m	<b>-(YNetwork*) nextNetwork</b>
pas	function <b>nextNetwork()</b> : TYNetwork
vb	function <b>nextNetwork()</b> As YNetwork
cs	<b>YNetwork nextNetwork()</b>
java	<b>YNetwork nextNetwork()</b>
py	<b>def 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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YNetworkValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YNetworkValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYNetworkValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**network→setAdminPassword()network→**  
**set\_adminPassword( )**

**YNetwork**

Changes the password for the "admin" user.

js	function <b>set_adminPassword( newval)</b>
node.js	function <b>set_adminPassword( newval)</b>
php	function <b>set_adminPassword( \$newval)</b>
cpp	int <b>set_adminPassword( const string&amp; newval)</b>
m	-(int) <b>setAdminPassword : (NSString*) newval</b>
pas	function <b>set_adminPassword( newval: string): integer</b>
vb	function <b>set_adminPassword( ByVal newval As String) As Integer</b>
cs	int <b>set_adminPassword( string newval)</b>
java	int <b>set_adminPassword( String newval)</b>
py	def <b>set_adminPassword( newval)</b>
cmd	YNetwork <b>target set_adminPassword 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( )**

**YNetwork**

Changes the credentials required to connect to the callback address.

<b>js</b>	function <b>set_callbackCredentials( newval)</b>
<b>nodejs</b>	function <b>set_callbackCredentials( newval)</b>
<b>php</b>	function <b>set_callbackCredentials( \$newval)</b>
<b>cpp</b>	int <b>set_callbackCredentials( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setCallbackCredentials : (NSString*) newval</b>
<b>pas</b>	function <b>set_callbackCredentials( newval: string): integer</b>
<b>vb</b>	function <b>set_callbackCredentials( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_callbackCredentials( string newval)</b>
<b>java</b>	int <b>set_callbackCredentials( String newval)</b>
<b>py</b>	def <b>set_callbackCredentials( newval)</b>
<b>cmd</b>	<b>YNetwork target set_callbackCredentials 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()**  
**network→setCallbackEncoding()network→**  
**set\_callbackEncoding( )**

**YNetwork**

Changes the encoding standard to use for representing notification values.

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

#### Parameters :

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

#### Returns :

`YAPI_SUCCESS` if the call succeeds.

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

**network→set\_callbackMaxDelay()** YNetwork  
**network→setCallbackMaxDelay()network→set\_callbackMaxDelay()**

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

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

**Parameters :**

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

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**network→set\_callbackMethod()**  
**network→setCallbackMethod()network→**  
**set\_callbackMethod( )**

**YNetwork**

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

<b>js</b>	function <b>set_callbackMethod( newval)</b>
<b>nodejs</b>	function <b>set_callbackMethod( newval)</b>
<b>php</b>	function <b>set_callbackMethod( \$newval)</b>
<b>cpp</b>	int <b>set_callbackMethod( Y_CALLBACKMETHOD_enum newval)</b>
<b>m</b>	-(int) <b>setCallbackMethod : (Y_CALLBACKMETHOD_enum) newval</b>
<b>pas</b>	function <b>set_callbackMethod( newval: Integer): integer</b>
<b>vb</b>	function <b>set_callbackMethod( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_callbackMethod( int newval)</b>
<b>java</b>	int <b>set_callbackMethod( int newval)</b>
<b>py</b>	def <b>set_callbackMethod( newval)</b>
<b>cmd</b>	<b>YNetwork target set_callbackMethod 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()**  
**network→setCallbackMinDelay()network→**  
**set\_callbackMinDelay()**

**YNetwork**

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

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

**Parameters :**

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

**Returns :**

YAPI\_SUCCESS if the call succeeds.

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

**network→set\_callbackUrl()**  
**network→setCallbackUrl()network→**  
**set\_callbackUrl( )**

**YNetwork**

Changes the callback URL to notify significant state changes.

<b>js</b>	function <b>set_callbackUrl( newval)</b>
<b>nodejs</b>	function <b>set_callbackUrl( newval)</b>
<b>php</b>	function <b>set_callbackUrl( \$newval)</b>
<b>cpp</b>	int <b>set_callbackUrl( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setCallbackUrl : (NSString*) newval</b>
<b>pas</b>	function <b>set_callbackUrl( newval: string): integer</b>
<b>vb</b>	function <b>set_callbackUrl( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_callbackUrl( string newval)</b>
<b>java</b>	int <b>set_callbackUrl( String newval)</b>
<b>py</b>	def <b>set_callbackUrl( newval)</b>
<b>cmd</b>	<b>YNetwork 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( )**

**YNetwork**

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

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

**Parameters :**

**newval** either **Y\_DISCOVERABLE\_FALSE** or **Y\_DISCOVERABLE\_TRUE**, according to the activation state of the multicast announce protocols to allow easy discovery of the module in the network neighborhood (uPnP/Bonjour protocol)

**Returns :**

**YAPI\_SUCCESS** if the call succeeds.

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

**network→set\_logicalName()**  
**network→setLogicalName()network→**  
**set\_logicalName( )**

**YNetwork**

Changes the logical name of the network interface.

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

**Returns :**

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

**network→set\_primaryDNS()**  
**network→setPrimaryDNS()** network→  
**set\_primaryDNS( )**

**YNetwork**

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

<b>js</b>	function <b>set_primaryDNS( newval)</b>
<b>nodejs</b>	function <b>set_primaryDNS( newval)</b>
<b>php</b>	function <b>set_primaryDNS( \$newval)</b>
<b>cpp</b>	int <b>set_primaryDNS( const string&amp; newval)</b>
<b>m</b>	- (int) <b>setPrimaryDNS : (NSString*) newval</b>
<b>pas</b>	function <b>set_primaryDNS( newval: string): integer</b>
<b>vb</b>	function <b>set_primaryDNS( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_primaryDNS( string newval)</b>
<b>java</b>	int <b>set_primaryDNS( String newval)</b>
<b>py</b>	def <b>set_primaryDNS( newval)</b>
<b>cmd</b>	<b>YNetwork target set_primaryDNS 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()**  
**network→setSecondaryDNS()network→**  
**set\_secondaryDNS( )**

**YNetwork**

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

<b>js</b>	function <b>set_secondaryDNS( newval)</b>
<b>node.js</b>	function <b>set_secondaryDNS( newval)</b>
<b>php</b>	function <b>set_secondaryDNS( \$newval)</b>
<b>cpp</b>	int <b>set_secondaryDNS( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setSecondaryDNS : (NSString*) newval</b>
<b>pas</b>	function <b>set_secondaryDNS( newval: string): integer</b>
<b>vb</b>	function <b>set_secondaryDNS( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_secondaryDNS( string newval)</b>
<b>java</b>	int <b>set_secondaryDNS( String newval)</b>
<b>py</b>	def <b>set_secondaryDNS( newval)</b>
<b>cmd</b>	<b>YNetwork 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)****YNetwork****network→setUserData()network→set(userData( )**

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

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

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

**Parameters :**

**data** any kind of object to be stored

**network→set\_userPassword()**  
**network→setUserPassword()network→**  
**set\_userPassword( )**

**YNetwork**

Changes the password for the "user" user.

<b>js</b>	function <b>set_userPassword( newval)</b>
<b>node.js</b>	function <b>set_userPassword( newval)</b>
<b>php</b>	function <b>set_userPassword( \$newval)</b>
<b>cpp</b>	int <b>set_userPassword( const string&amp; newval)</b>
<b>m</b>	- <b>(int setUserPassword : (NSString*) newval</b>
<b>pas</b>	function <b>set_userPassword( newval: string): integer</b>
<b>vb</b>	function <b>set_userPassword( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_userPassword( string newval)</b>
<b>java</b>	int <b>set_userPassword( String newval)</b>
<b>py</b>	def <b>set_userPassword( newval)</b>
<b>cmd</b>	<b>YNetwork target set_userPassword 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.

<code>js</code>	function <b>set_wwwWatchdogDelay( newval)</b>
<code>nodejs</code>	function <b>set_wwwWatchdogDelay( newval)</b>
<code>php</code>	function <b>set_wwwWatchdogDelay( \$newval)</b>
<code>cpp</code>	int <b>set_wwwWatchdogDelay( int newval)</b>
<code>m</code>	- <b>(int) setWwwWatchdogDelay : (int) newval</b>
<code>pas</code>	function <b>set_wwwWatchdogDelay( newval: LongInt): integer</b>
<code>vb</code>	function <b>set_wwwWatchdogDelay( ByVal newval As Integer) As Integer</b>
<code>cs</code>	int <b>set_wwwWatchdogDelay( int newval)</b>
<code>java</code>	int <b>set_wwwWatchdogDelay( int newval)</b>
<code>py</code>	def <b>set_wwwWatchdogDelay( newval)</b>
<code>cmd</code>	YNetwork <b>target set_wwwWatchdogDelay 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()****YNetwork**

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

```

js function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
nodejs function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
php function useDHCP( $fallbackIpAddr, $fallbackSubnetMaskLen, $fallbackRouter)
cpp int useDHCP( string fallbackIpAddr,
                  int fallbackSubnetMaskLen,
                  string fallbackRouter)

m -(int) useDHCP : (NSString*) fallbackIpAddr
                  : (int) fallbackSubnetMaskLen
                  : (NSString*) fallbackRouter

pas function useDHCP( fallbackIpAddr: string,
                      fallbackSubnetMaskLen: LongInt,
                      fallbackRouter: string): integer

vb function useDHCP( ByVal fallbackIpAddr As String,
                     ByVal fallbackSubnetMaskLen As Integer,
                     ByVal fallbackRouter As String) As Integer

cs int useDHCP( string fallbackIpAddr,
                 int fallbackSubnetMaskLen,
                 string fallbackRouter)

java int useDHCP( String fallbackIpAddr,
                  int fallbackSubnetMaskLen,
                  String fallbackRouter)

py def useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
cmd YNetwork target useDHCP fallbackIpAddr fallbackSubnetMaskLen fallbackRouter

```

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

**Parameters :**

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

**Returns :**

`YAPI_SUCCESS` if the call succeeds.

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

**network→useStaticIP()****YNetwork**

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

```

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

```

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

**Parameters :**

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

**Returns :**

`YAPI_SUCCESS` if the call succeeds.

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

## network→wait\_async()

## YNetwork

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

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

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

### Parameters :

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

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

### Returns :

nothing.

## 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');
cpp	#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)**

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() yFindOsControl()yFindOsControl( )

YOsControl

Retrieves OS control for a given identifier.

js	function <b>yFindOsControl( func)</b>
node.js	function <b>FindOsControl( func)</b>
php	function <b>yFindOsControl( \$func)</b>
cpp	<b>YOsControl*</b> <b>yFindOsControl( const string&amp; func)</b>
m	<b>YOsControl*</b> <b>yFindOsControl( NSString* func)</b>
pas	function <b>yFindOsControl( func: string): TYOsControl</b>
vb	function <b>yFindOsControl( ByVal func As String) As YOsControl</b>
cs	<b>YOsControl</b> <b>FindOsControl( string func)</b>
java	<b>YOsControl</b> <b>FindOsControl( String func)</b>
py	def <b>FindOsControl( 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()yFirstOsControl( )**

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()****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	- <b>(NSString*) describe</b>
pas	function <b>describe( ): string</b>
vb	function <b>describe( ) As String</b>
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**oscontrol→get\_advertisedValue()**  
**oscontrol→advertisedValue()oscontrol→**  
**get\_advertisedValue( )**

**YOsControl**

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

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	YOsControl <b>target 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()**  
**oscontrol→errorMessage()**  
**oscontrol→get\_errorMessage( )****YOsControl**

Returns the error message of the latest error with the OS control.

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

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

**Returns :**

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

**oscontrol→get\_errorType()**  
**oscontrol→errorType()oscontrol→**  
**get\_errorType( )**

**YOscControl**

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	<b>YRETCODE get_errorType( )</b>
pas	function <b>get_errorType( ): YRETCODE</b>
vb	function <b>get_errorType( ) As YRETCODE</b>
cs	<b>YRETCODE 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()**  
**oscontrol→friendlyName()** oscontrol→  
**get\_friendlyName( )**

**YOsControl**

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

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

The returned string uses the logical names of the module and of the 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>
node.js	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.

**oscontrol→get\_functionId()**  
**oscontrol→functionId()** oscontrol→  
**get\_functionId( )**

**YOsControl**

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()**  
**oscontrol→hardwareId()** oscontrol→  
**get\_hardwareId( )**

**YOsControl**

Returns the unique hardware identifier of the OS control in the form SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the OS control (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**oscontrol→get\_logicalName()**  
**oscontrol→logicalName()**  
**oscontrol→get\_logicalName( )**

**YOsControl**

Returns the logical name of the OS control.

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

**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>
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 online.

**Returns :**

an instance of YModule

**oscontrol→get\_module\_async()**  
**oscontrol→module\_async()****YOsControl**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**oscontrol→get\_shutdownCountdown()**  
**oscontrol→shutdownCountdown()oscontrol→**  
**get\_shutdownCountdown( )**

**YOsControl**

Returns the remaining number of seconds before the OS shutdown, or zero when no shutdown has been scheduled.

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

**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> ( )
node.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.

**oscontrol→isOnline()****YOsControl**

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

js	function <b>isOnline( )</b>
node.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 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

## oscontrol→isOnline\_async()

YOsControl

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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</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.

**oscontrol→load\_async()****YOsControl**

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( ): TYOsControl</b>
vb	function <b>nextOsControl( ) As YOsControl</b>
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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YOsControlValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YOsControlValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYOsControlValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**oscontrol→setLogicalName()**  
**oscontrol→set\_logicalName( )**

**YOsControl**

Changes the logical name of the OS control.

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

**Returns :**

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

**oscontrol→set(userData)**  
**oscontrol→setUserData()****oscontrol→set(userData)**

**YOsControl**

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

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData)</b>
vb	procedure <b>set(userData)</b> ByVal <b>data As Object</b>
cs	void <b>set(userData)</b>
java	void <b>set(userData)</b>
py	def <b>set(userData)</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()****YOControl**

Schedules an OS shutdown after a given number of seconds.

```
js function shutdown( secBeforeShutDown)
nodejs function shutdown( secBeforeShutDown)
php function shutdown( $secBeforeShutDown)
cpp int shutdown( int secBeforeShutDown)
m -(int) shutdown : (int) secBeforeShutDown
pas function shutdown( secBeforeShutDown: LongInt): LongInt
vb function shutdown( ) As Integer
cs int shutdown( int secBeforeShutDown)
java int shutdown( int secBeforeShutDown)
py def shutdown( secBeforeShutDown)
cmd YOControl target shutdown secBeforeShutDown
```

**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');
cpp	#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()

### 3. Reference

Returns the hardware identifier of the electrical power sensor, without reference to the module.
<b>power→get_hardwareId()</b>
Returns the unique hardware identifier of the electrical power sensor in the form SERIAL.FUNCTIONID.
<b>power→get_highestValue()</b>
Returns the maximal value observed for the electrical power.
<b>power→get_logFrequency()</b>
Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>power→get_logicalName()</b>
Returns the logical name of the electrical power sensor.
<b>power→get_lowestValue()</b>
Returns the minimal value observed for the electrical power.
<b>power→get_meter()</b>
Returns the energy counter, maintained by the wattmeter by integrating the power consumption over time.
<b>power→get_meterTimer()</b>
Returns the elapsed time since last energy counter reset, in seconds.
<b>power→get_module()</b>
Gets the YModule object for the device on which the function is located.
<b>power→get_module_async(callback, context)</b>
Gets the YModule object for the device on which the function is located (asynchronous version).
<b>power→get_recordedData(startTime, endTime)</b>
Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>power→get_reportFrequency()</b>
Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>power→get_resolution()</b>
Returns the resolution of the measured values.
<b>power→get_unit()</b>
Returns the measuring unit for the electrical power.
<b>power→get(userData)</b>
Returns the value of the userData attribute, as previously stored using method set(userData).
<b>power→isOnline()</b>
Checks if the electrical power sensor is currently reachable, without raising any error.
<b>power→isOnline_async(callback, context)</b>
Checks if the electrical power sensor is currently reachable, without raising any error (asynchronous version).
<b>power→load(msValidity)</b>
Preloads the electrical power sensor cache with a specified validity duration.
<b>power→loadCalibrationPoints(rawValues, refValues)</b>
Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>power→load_async(msValidity, callback, context)</b>
Preloads the electrical power sensor cache with a specified validity duration (asynchronous version).
<b>power→nextPower()</b>
Continues the enumeration of electrical power sensors started using yFirstPower( ).
<b>power→registerTimedReportCallback(callback)</b>
Registers the callback function that is invoked on every periodic timed notification.
<b>power→registerValueCallback(callback)</b>

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() yFindPower()yFindPower( )

YPower

Retrieves a electrical power sensor for a given identifier.

js	function <b>yFindPower( func)</b>
node.js	function <b>FindPower( func)</b>
php	function <b>yFindPower( \$func)</b>
cpp	<b>YPower*</b> <b>yFindPower( const string&amp; func)</b>
m	<b>YPower*</b> <b>yFindPower( NSString* func)</b>
pas	function <b>yFindPower( func: string): TYPower</b>
vb	function <b>yFindPower( ByVal func As String) As YPower</b>
cs	<b>YPower FindPower( string func)</b>
java	<b>YPower FindPower( String func)</b>
py	def <b>FindPower( 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()yFirstPower( )

## YPower

Starts the enumeration of electrical power sensors currently accessible.

```
js function yFirstPower( )
nodejs function FirstPower( )
php function yFirstPower( )
cpp YPower* yFirstPower( )
m YPower* yFirstPower( )
pas function yFirstPower( ): TYPower
vb function yFirstPower( ) As YPower
cs YPower FirstPower( )
java YPower FirstPower( )
py def FirstPower( )
```

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()****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)=RELAYL01-123456.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 electrical power sensor (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**power→get\_advertisedValue()**  
**power→advertisedValue()** power→  
**get\_advertisedValue( )**

YPower

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

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

**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	<b>double get_cosPhi( )</b>
m	-(double) cosPhi
pas	function <b>get_cosPhi( ): double</b>
vb	function <b>get_cosPhi( ) As Double</b>
cs	<b>double get_cosPhi( )</b>
java	<b>double get_cosPhi( )</b>
py	<b>def get_cosPhi( )</b>
cmd	<b>YPower target 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()**  
**power→currentRawValue()power→**  
**get\_currentRawValue( )**

YPower

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

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( )**: double  
**vb** function **get\_currentRawValue( )** As Double  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YPower **target get\_currentRawValue**

**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()**  
**power→currentValue()** power→  
**get\_currentValue()**

YPower

Returns the current measure for the electrical power.

js	function <b>get_currentValue( )</b>
node.js	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 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→getErrorMessage()**  
**power→errorMessage()** power→  
**getErrorMessage( )**

YPower

Returns the error message of the latest error with the electrical power sensor.

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

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()`  
`power→friendlyName()power→`  
`get_friendlyName( )`

**YPower**

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

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

The returned string uses the logical names of the module and of the electrical power sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the electrical power sensor (for exemple: 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()**  
**power→functionDescriptor()power→**  
**get\_functionDescriptor( )**

**YPower**

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

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	<b>String get_functionDescriptor( )</b>
<b>py</b>	<b>def 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>
node.js	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>
node.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( ) As String</b>
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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the electrical power sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**power→get\_highestValue()**  
**power→highestValue()** power→  
**get\_highestValue( )**

YPower

Returns the maximal value observed for the electrical power.

```
js function get_highestValue( )
nodejs function get_highestValue( )
php function get_highestValue( )
cpp double get_highestValue( )
m -(double) highestValue
pas function get_highestValue( ): double
vb function get_highestValue( ) As Double
cs double get_highestValue( )
java double get_highestValue( )
py def get_highestValue( )
cmd YPower target get_highestValue
```

**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()**  
**power→logFrequency()power→**  
**get\_logFrequency( )**

YPower

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

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

**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>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	- <b>(NSString*) logicalName</b>
pas	function <b>get_logicalName( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	<b>String get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	<b>YPower target 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 get_lowestValue( )  
nodejs function get_lowestValue( )  
php  function get_lowestValue( )  
cpp   double get_lowestValue( )  
m    -(double) lowestValue  
pas   function get_lowestValue( ): double  
vb    function get_lowestValue( ) As Double  
cs    double get_lowestValue( )  
java  double get_lowestValue( )  
py    def get_lowestValue( )  
cmd   YPower target get_lowestValue
```

**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()**  
**power→meter()power→get\_meter( )****YPower**

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( ): double</b>
vb	function <b>get_meter( ) As Double</b>
cs	double <b>get_meter( )</b>
java	double <b>get_meter( )</b>
py	<b>def get_meter( )</b>
cmd	<b>YPower target 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	<b>YPower target 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()**  
**power→module()power→get\_module()****YPower**

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

js	<code>function get_module( )</code>
node.js	<code>function get_module( )</code>
php	<code>function get_module( )</code>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	<code>function get_module( ): TYModule</code>
vb	<code>function get_module( ) As YModule</code>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

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

**Returns :**

an instance of `YModule`

## power→get\_module\_async() power→module\_async()

YPower

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

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

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

### Returns :

nothing : the result is provided to the callback.

**power→get\_recordedData()**  
**power→recordedData()power→**  
**get\_recordedData( )**

**YPower**

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

js	function <b>get_recordedData( startTime, endTime)</b>
nodejs	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	YDataSet <b>get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
java	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
py	def <b>get_recordedData( startTime, endTime)</b>
cmd	<b>YPower target get_recordedData startTime 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 measure, 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 measure, 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()**  
**power→reportFrequency()power→**  
**get\_reportFrequency( )**

YPower

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

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

**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>
node.js	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 target <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*) <b>unit</b>
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	<b>YPower target 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)**

YPower

**power→userData() power→get(userData)()**

Returns the value of the userData attribute, as previously stored using method set(userData).

js	function get(userData) {
node.js	function get(userData) {
php	function get(userData) {
cpp	void * get(userData) {
m	-(void*) userData
pas	function get(userData): TObject;
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData):

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

**Returns :**

the object stored previously by the caller.

**power→isOnline()****YPower**

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

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) 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	<b>def 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()****YPower**

Preloads the electrical power sensor cache with a specified validity duration.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</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.

**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()****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 registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YPowerTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YPowerTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYPowerTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YPowerValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YPowerValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYPowerValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()****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	- <b>(int) reset</b>
pas	function <b>reset( ): LongInt</b>
vb	function <b>reset( ) As Integer</b>
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()**  
**power→setHighestValue()** power→  
**set\_highestValue( )**

YPower

Changes the recorded maximal value observed pour the electrical power.

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

#### 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()**  
**power→setLogFrequency()power→**  
**set\_logFrequency( )**

YPower

---

Changes the datalogger recording frequency for this function.

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

**power→set\_logicalName()**  
**power→setLogicalName()** power→  
**set\_logicalName( )**

YPower

Changes the logical name of the electrical power sensor.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the electrical power sensor.

**Returns :**

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

**power→set\_lowestValue()**  
**power→setLowestValue()power→**  
**set\_lowestValue( )**

YPower

Changes the recorded minimal value observed pour the electrical power.

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

**power→set\_resolution()**

YPower

**power→setResolution()power→set\_resolution()**

Changes the resolution of the measured values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YPower target set_resolution newval
```

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)**

YPower

**power→setUserData() power→set(userData())**

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

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

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

**Parameters :**

**data** any kind of object to be stored

## 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');
cpp #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)**

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()****YPressure****yFindPressure()yFindPressure( )**

Retrieves a pressure sensor for a given identifier.

<code>js</code>	<code>function yFindPressure( func)</code>
<code>node.js</code>	<code>function FindPressure( func)</code>
<code>php</code>	<code>function yFindPressure( \$func)</code>
<code>cpp</code>	<code>YPressure* yFindPressure( const string&amp; func)</code>
<code>m</code>	<code>YPressure* yFindPressure( NSString* func)</code>
<code>pas</code>	<code>function yFindPressure( func: string): TYPressure</code>
<code>vb</code>	<code>function yFindPressure( ByVal func As String) As YPressure</code>
<code>cs</code>	<code>YPressure FindPressure( string func)</code>
<code>java</code>	<code>YPressure FindPressure( String func)</code>
<code>py</code>	<code>def FindPressure( 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 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()yFirstPressure( )**

Starts the enumeration of pressure sensors currently accessible.

js	function <b>yFirstPressure( )</b>
node.js	function <b>FirstPressure( )</b>
php	function <b>yFirstPressure( )</b>
cpp	YPressure* <b>yFirstPressure( )</b>
m	YPressure* <b>yFirstPressure( )</b>
pas	function <b>yFirstPressure( )</b> : TYPressure
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 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  YPressure 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.

**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 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**pressure→get\_advertisedValue()**  
**pressure→advertisedValue()pressure→**  
**get\_advertisedValue( )**

**YPressure**

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

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>node.js</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YPressure target 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()**  
**pressure→currentRawValue()****pressure→get\_currentRawValue( )**

**YPressure**

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

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( )**: double  
**vb** function **get\_currentRawValue( )** As Double  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YPressure target **get\_currentRawValue**

**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()**  
**pressure→currentValue()pressure→**  
**get\_currentValue()**

**YPressure**

Returns the current measure for the pressure.

<b>js</b>	function <b>get_currentValue( )</b>
<b>node.js</b>	function <b>get_currentValue( )</b>
<b>php</b>	function <b>get_currentValue( )</b>
<b>cpp</b>	double <b>get_currentValue( )</b>
<b>m</b>	-(double) <b>currentValue</b>
<b>pas</b>	function <b>get_currentValue( )</b> : double
<b>vb</b>	function <b>get_currentValue( )</b> As Double
<b>cs</b>	double <b>get_currentValue( )</b>
<b>java</b>	double <b>get_currentValue( )</b>
<b>py</b>	def <b>get_currentValue( )</b>
<b>cmd</b>	<b>YPressure target 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()**  
**pressure→errorMessage()** pressure→  
**get\_errorMessage( )**

**YPressure**

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

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

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

**Returns :**

a string corresponding to the latest error message that 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()**  
**pressure→friendlyName()pressure→**  
**get\_friendlyName( )**

**YPressure**

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

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

The returned string uses the logical names of the module and of the 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>
node.js	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.

**pressure→get\_functionId()**  
**pressure→functionId()** **pressure→get\_functionId( )**

**YPressure**

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>
node.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( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	<b>String get_hardwareId( )</b>
py	<b>def 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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the pressure sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**pressure→get\_highestValue()**  
**pressure→highestValue()** pressure→  
**get\_highestValue( )**

**YPressure**

Returns the maximal value observed for the pressure.

```
js function get_highestValue( )
nodejs function get_highestValue( )
php function get_highestValue( )
cpp double get_highestValue( )
m -(double) highestValue
pas function get_highestValue( ): double
vb function get_highestValue( ) As Double
cs double get_highestValue( )
java double get_highestValue( )
py def get_highestValue( )
cmd YPressure target get_highestValue
```

**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()**  
**pressure→logFrequency()pressure→**  
**get\_logFrequency( )**

**YPressure**

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

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

**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()**  
**pressure→logicalName()pressure→**  
**get\_logicalName( )**

**YPressure**

Returns the logical name of the pressure sensor.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	<b>YPressure target 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()**  
**pressure→lowestValue()pressure→**  
**get\_lowestValue( )**

**YPressure**

Returns the minimal value observed for the pressure.

<b>js</b>	function <b>get_lowestValue( )</b>
<b>nodejs</b>	function <b>get_lowestValue( )</b>
<b>php</b>	function <b>get_lowestValue( )</b>
<b>cpp</b>	double <b>get_lowestValue( )</b>
<b>m</b>	-(double) lowestValue
<b>pas</b>	function <b>get_lowestValue( ): double</b>
<b>vb</b>	function <b>get_lowestValue( ) As Double</b>
<b>cs</b>	double <b>get_lowestValue( )</b>
<b>java</b>	double <b>get_lowestValue( )</b>
<b>py</b>	def <b>get_lowestValue( )</b>
<b>cmd</b>	<b>YPressure target 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>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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

**Returns :**

an instance of `YModule`

**pressure→get\_module\_async()**  
**pressure→module\_async()****YPressure**

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

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

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

```
pressure→get_recordedData()  
pressure→recordedData()pressure→  
get_recordedData()
```

## YPressure

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

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 measurement, 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 measure, 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()**  
**pressure→reportFrequency()pressure→**  
**get\_reportFrequency( )**

**YPressure**

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

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

**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 get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YPressure target get_resolution
```

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*) <b>unit</b>
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 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 get(userData) 
node.js function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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

**Returns :**

the object stored previously by the caller.

**pressure→isOnline()****YPressure**

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

js	function <b>isOnline( )</b>
node.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 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()****YPressure**

Preloads the pressure sensor cache with a specified validity duration.

js	function <b>load( msValidity)</b>
node.js	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	<b>YRETCODE load( int msValidity)</b>
m	<b>-(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	<b>YRETCODE load( int msValidity)</b>
java	<b>int load( long msValidity)</b>
py	<b>def load( 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 <b>loadCalibrationPoints( rawValues, refValues )</b>
nodejs	function <b>loadCalibrationPoints( rawValues, refValues )</b>
php	function <b>loadCalibrationPoints( &amp;\$rawValues, &amp;\$refValues )</b>
cpp	int <b>loadCalibrationPoints( vector&lt;double&gt;&amp; rawValues, vector&lt;double&gt;&amp; refValues )</b>
m	- <b>(int) loadCalibrationPoints : (NSMutableArray*) rawValues : (NSMutableArray*) refValues</b>
pas	function <b>loadCalibrationPoints( var rawValues: TDoubleArray, var refValues: TDoubleArray): LongInt</b>
vb	procedure <b>loadCalibrationPoints( )</b>
cs	int <b>loadCalibrationPoints( List&lt;double&gt; rawValues, List&lt;double&gt; refValues )</b>
java	int <b>loadCalibrationPoints( ArrayList&lt;Double&gt; rawValues, ArrayList&lt;Double&gt; refValues )</b>
py	def <b>loadCalibrationPoints( rawValues, refValues )</b>
cmd	YPressure target <b>loadCalibrationPoints rawValues refValues</b>

**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 **nextPressure( )**  
`node.js` function **nextPressure( )**  
`php` function **nextPressure( )**  
`cpp` **YPressure \* nextPressure( )**  
`m` -(**YPressure\***) **nextPressure**  
`pas` function **nextPressure( )**: TYPressure  
`vb` function **nextPressure( )** As YPressure  
`cs` **YPressure nextPressure( )**  
`java` **YPressure nextPressure( )**  
`py` def **nextPressure( )**

**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.

<b>js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>node.js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>php</b>	function registerTimedReportCallback( <b>\$callback</b> )
<b>cpp</b>	int registerTimedReportCallback( YPressureTimedReportCallback <b>callback</b> )
<b>m</b>	-(int) registerTimedReportCallback : (YPressureTimedReportCallback) <b>callback</b>
<b>pas</b>	function registerTimedReportCallback( <b>callback</b> : TYPressureTimedReportCallback): LongInt
<b>vb</b>	function registerTimedReportCallback( ) As Integer
<b>cs</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>java</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>py</b>	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YPressureValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YPressureValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYPRESSUREValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**pressure→setHighestValue()** **pressure→set\_highestValue()**

**YPressure**

Changes the recorded maximal value observed for the pressure.

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

**YPressure**

Changes the datalogger recording frequency for this function.

```
js   function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php  function set_logFrequency( $newval)
cpp  int set_logFrequency( const string& newval)
m    -(int) setLogFrequency : (NSString*) newval
pas   function set_logFrequency( newval: string): integer
vb   function set_logFrequency( ByVal newval As String) As Integer
cs   int set_logFrequency( string newval)
java  int set_logFrequency( String newval)
py    def set_logFrequency( newval)
cmd  YPressure target set_logFrequency 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 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()**  
**pressure→setLogicalName()** **pressure→**  
**set\_logicalName( )**

**YPressure**

Changes the logical name of the pressure sensor.

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

**Returns :**

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

**pressure→set\_lowestValue()**  
**pressure→setLowestValue()pressure→**  
**set\_lowestValue()**

**YPressure**

Changes the recorded minimal value observed for the pressure.

```
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 YPressure target set_lowestValue newval
```

**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.

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

**pressure→set\_resolution()**  
**pressure→setResolution()****pressure→set\_resolution()**

**YPressure**

Changes the resolution of the measured physical values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YPressure target set_resolution newval
```

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)**  
**pressure→setUserData()pressure→**  
**set(userData( )**

**YPressure**

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

js	function <b>set(userData( data)</b>
node.js	function <b>set(userData( data)</b>
php	function <b>set(userData( \$data)</b>
cpp	void <b>set(userData( void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData( data: Tobject)</b>
vb	procedure <b>set(userData( ByVal data As Object)</b>
cs	void <b>set(userData( object data)</b>
java	void <b>set(userData( Object data)</b>
py	def <b>set(userData( 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>
node.js	var yoctolib = require('yoctolib');
	var YPwmOutput = yoctolib.YPwmOutput;
php	require_once('yocto_pwmoutput.php');
cpp	#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 cyle as a floating point number between 0 an 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()

### 3. Reference

Returns a unique identifier of type YFUN_DESCR corresponding to the function.
<b>pwmoutput-&gt;get_functionId()</b> Returns the hardware identifier of the PWM, without reference to the module.
<b>pwmoutput-&gt;get_hardwareId()</b> Returns the unique hardware identifier of the PWM in the form SERIAL.FUNCTIONID.
<b>pwmoutput-&gt;get_logicalName()</b> Returns the logical name of the PWM.
<b>pwmoutput-&gt;get_module()</b> Gets the YModule object for the device on which the function is located.
<b>pwmoutput-&gt;get_module_async(callback, context)</b> Gets the YModule object for the device on which the function is located (asynchronous version).
<b>pwmoutput-&gt;get_period()</b> Returns the PWM period in nanoseconds.
<b>pwmoutput-&gt;get_pulseDuration()</b> Returns the PWM pulse length in milliseconds.
<b>pwmoutput-&gt;get_userData()</b> Returns the value of the userData attribute, as previously stored using method set(userData).
<b>pwmoutput-&gt;isOnline()</b> Checks if the PWM is currently reachable, without raising any error.
<b>pwmoutput-&gt;isOnline_async(callback, context)</b> Checks if the PWM is currently reachable, without raising any error (asynchronous version).
<b>pwmoutput-&gt;load(msValidity)</b> Preloads the PWM cache with a specified validity duration.
<b>pwmoutput-&gt;load_async(msValidity, callback, context)</b> Preloads the PWM cache with a specified validity duration (asynchronous version).
<b>pwmoutput-&gt;nextPwmOutput()</b> Continues the enumeration of PWMs started using yFirstPwmOutput( ).
<b>pwmoutput-&gt;pulseDurationMove(ms_target, ms_duration)</b> Performs a smooth change of the pulse duration toward a given value.
<b>pwmoutput-&gt;registerValueCallback(callback)</b> Registers the callback function that is invoked on every change of advertised value.
<b>pwmoutput-&gt;set_dutyCycle(newval)</b> Configures the PWMs duty cycle.
<b>pwmoutput-&gt;set_dutyCycleAtPowerOn(newval)</b> Configures the PWMs duty cycle at device power up.
<b>pwmoutput-&gt;set_enabled(newval)</b> Stops or starts the PWM.
<b>pwmoutput-&gt;set_enabledAtPowerOn(newval)</b> Configures the state of PWM at device power up.
<b>pwmoutput-&gt;set_frequency(newval)</b> Configures the PWM frequency.
<b>pwmoutput-&gt;set_logicalName(newval)</b> Changes the logical name of the PWM.
<b>pwmoutput-&gt;set_period(newval)</b> Configures the PWM period.

**pwmoutput→set\_pulseDuration(newval)**

Configures the PWM pulses length.

**pwmoutput→set(userData)**

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() yFindPwmOutput()yFindPwmOutput( )

**YPwmOutput**

Retrieves a PWM for a given identifier.

js	function <b>yFindPwmOutput( func)</b>
node.js	function <b>FindPwmOutput( func)</b>
php	function <b>yFindPwmOutput( \$func)</b>
cpp	YPwmOutput* <b>yFindPwmOutput( const string&amp; func)</b>
m	YPwmOutput* <b>yFindPwmOutput( NSString* func)</b>
pas	function <b>yFindPwmOutput( func: string): TYPwmOutput</b>
vb	function <b>yFindPwmOutput( ByVal func As String) As YPwmOutput</b>
cs	YPwmOutput <b>FindPwmOutput( string func)</b>
java	YPwmOutput <b>FindPwmOutput( String func)</b>
py	def <b>FindPwmOutput( 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()****yFirstPwmOutput()yFirstPwmOutput( )****YPwmOutput**

Starts the enumeration of PWMs currently accessible.

js	function <b>yFirstPwmOutput( )</b>
node.js	function <b>FirstPwmOutput( )</b>
php	function <b>yFirstPwmOutput( )</b>
cpp	YPwmOutput* <b>yFirstPwmOutput( )</b>
m	YPwmOutput* <b>yFirstPwmOutput( )</b>
pas	function <b>yFirstPwmOutput( ): TYPwmOutput</b>
vb	function <b>yFirstPwmOutput( ) As YPwmOutput</b>
cs	YPwmOutput <b>FirstPwmOutput( )</b>
java	YPwmOutput <b>FirstPwmOutput( )</b>
py	def <b>FirstPwmOutput( )</b>

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()****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)=RELAYL01-123456.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 PWM (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pwmoutput→dutyCycleMove()**  
**pwmoutput→dutyCycleMove( )**

**YPwmOutput**

Performs a smooth change of the pulse duration toward a given value.

<b>js</b>	function <b>dutyCycleMove( target, ms_duration)</b>
<b>nodejs</b>	function <b>dutyCycleMove( target, ms_duration)</b>
<b>php</b>	function <b>dutyCycleMove( \$target, \$ms_duration)</b>
<b>cpp</b>	int <b>dutyCycleMove( double target, int ms_duration)</b>
<b>m</b>	-(int) <b>dutyCycleMove : (double) target : (int) ms_duration</b>
<b>pas</b>	function <b>dutyCycleMove( target: double, ms_duration: LongInt): LongInt</b>
<b>vb</b>	function <b>dutyCycleMove( ) As Integer</b>
<b>cs</b>	int <b>dutyCycleMove( double target, int ms_duration)</b>
<b>java</b>	int <b>dutyCycleMove( double target, int ms_duration)</b>
<b>py</b>	def <b>dutyCycleMove( target, ms_duration)</b>
<b>cmd</b>	<b>YPwmOutput target dutyCycleMove target 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()`  
`pwmoutput->advertisedValue()`  
`pwmoutput->get_advertisedValue( )`

`YPwmOutput`

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

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

**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()**  
**pwmoutput→dutyCycle()****pwmoutput→get\_dutyCycle( )**

**YPwmOutput**

Returns the PWMs dutty cyle as a floating point number between 0 an 1.

js	function <b>get_dutyCycle( )</b>
node.js	function <b>get_dutyCycle( )</b>
php	function <b>get_dutyCycle( )</b>
cpp	double <b>get_dutyCycle( )</b>
m	-(double) <b>dutyCycle</b>
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 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 get_dutyCycleAtPowerOn( )`  
`nodejs` `function get_dutyCycleAtPowerOn( )`  
`php` `function get_dutyCycleAtPowerOn( )`  
`cpp` `double get_dutyCycleAtPowerOn( )`  
`m` `-(double) dutyCycleAtPowerOn`  
`pas` `function get_dutyCycleAtPowerOn( ): double`  
`vb` `function get_dutyCycleAtPowerOn( ) As Double`  
`cs` `double get_dutyCycleAtPowerOn( )`  
`java` `double get_dutyCycleAtPowerOn( )`  
`py` `def get_dutyCycleAtPowerOn( )`  
`cmd` `YPwmOutput target get_dutyCycleAtPowerOn`

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 target <b>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()`  
`pwmoutput->enabledAtPowerOn()``pwmoutput->`  
`get_enabledAtPowerOn( )`

`YPwmOutput`

Returns the state of the PWMs at device power up.

<code>js</code>	<code>function get_enabledAtPowerOn( )</code>
<code>nodejs</code>	<code>function get_enabledAtPowerOn( )</code>
<code>php</code>	<code>function get_enabledAtPowerOn( )</code>
<code>cpp</code>	<code>Y_ENABLEDATPOWERON_enum get_enabledAtPowerOn( )</code>
<code>m</code>	<code>-(Y_ENABLEDATPOWERON_enum) enabledAtPowerOn</code>
<code>pas</code>	<code>function get_enabledAtPowerOn( ): Integer</code>
<code>vb</code>	<code>function get_enabledAtPowerOn( ) As Integer</code>
<code>cs</code>	<code>int get_enabledAtPowerOn( )</code>
<code>java</code>	<code>int get_enabledAtPowerOn( )</code>
<code>py</code>	<code>def get_enabledAtPowerOn( )</code>
<code>cmd</code>	<code>YPwmOutput target get_enabledAtPowerOn</code>

**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()**  
**pwmoutput→errorMessage()****pwmoutput→**  
**get\_errorMessage( )**

**YPwmOutput**

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

<b>js</b>	function <b>get_errorMessage( )</b>
<b>node.js</b>	function <b>get_errorMessage( )</b>
<b>php</b>	function <b>get_errorMessage( )</b>
<b>cpp</b>	string <b>get_errorMessage( )</b>
<b>m</b>	-(NSString*) errorMessage
<b>pas</b>	function <b>get_errorMessage( )</b> : string
<b>vb</b>	function <b>get_errorMessage( )</b> As String
<b>cs</b>	string <b>get_errorMessage( )</b>
<b>java</b>	String <b>get_errorMessage( )</b>
<b>py</b>	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()**  
**pwmoutput→errorType()**  
**pwmoutput→get\_errorType( )**

**YPwmOutput**

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

**js** function **get\_errorType( )**  
**nodejs** function **get\_errorType( )**  
**php** function **get\_errorType( )**  
**cpp** YRETCODE **get\_errorType( )**  
**pas** function **get\_errorType( )**: YRETCODE  
**vb** function **get\_errorType( )** As YRETCODE  
**cs** YRETCODE **get\_errorType( )**  
**java** int **get\_errorType( )**  
**py** def **get\_errorType( )**

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the PWM object

**pwmoutput→get\_frequency()**  
**pwmoutput→frequency()****pwmoutput→get\_frequency( )**

**YPwmOutput**

Returns the PWM frequency in Hz.

<b>js</b>	function <b>get_frequency( )</b>
<b>nodejs</b>	function <b>get_frequency( )</b>
<b>php</b>	function <b>get_frequency( )</b>
<b>cpp</b>	int <b>get_frequency( )</b>
<b>m</b>	-(int) <b>frequency</b>
<b>pas</b>	function <b>get_frequency( )</b> : LongInt
<b>vb</b>	function <b>get_frequency( )</b> As Integer
<b>cs</b>	int <b>get_frequency( )</b>
<b>java</b>	int <b>get_frequency( )</b>
<b>py</b>	def <b>get_frequency( )</b>
<b>cmd</b>	<b>YPwmOutput target 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()**  
**pwmoutput→friendlyName()****pwmoutput→get\_friendlyName( )**

**YPwmOutput**

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

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

The returned string uses the logical names of the module and of the 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>
node.js	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.

**pwmoutput→get\_functionId()**  
**pwmoutput→functionId()** **pwmoutput→**  
**get\_functionId( )**

**YPwmOutput**

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*) <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 PWM (ex: `relay1`) On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

**pwmoutput→get\_hardwareId()**  
**pwmoutput→hardwareId()****pwmoutput→get\_hardwareId( )**

**YPwmOutput**

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

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the PWM. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the PWM (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**pwmoutput→get\_logicalName()**  
**pwmoutput→logicalName()****pwmoutput→get\_logicalName( )**

**YPwmOutput**

Returns the logical name of the PWM.

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

**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 get_module_async( callback, context)`  
`node.js` `function get_module_async( callback, context)`

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**pwmoutput→get\_period()****YPwmOutput****pwmoutput→period()pwmoutput→get\_period( )**

Returns the PWM period in nonaseconde.

js	function <b>get_period( )</b>
node.js	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 target <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 get_pulseDuration( )`  
`nodejs` `function get_pulseDuration( )`  
`php` `function get_pulseDuration( )`  
`cpp` `double get_pulseDuration( )`  
`m` `-(double) pulseDuration`  
`pas` `function get_pulseDuration( ): double`  
`vb` `function get_pulseDuration( ) As Double`  
`cs` `double get_pulseDuration( )`  
`java` `double get_pulseDuration( )`  
`py` `def get_pulseDuration( )`  
`cmd` `YPwmOutput target get_pulseDuration`

**Returns :**

a floating point number corresponding to the PWM pulse length in milliseconds

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

**pwmoutput→get(userData)**  
**pwmoutput→userData()****pwmoutput→get(userData)**

**YPwmOutput**

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()****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	- <b>(BOOL) 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()****YPwmOutput**

Preloads the PWM cache with a specified validity duration.

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

<code>js</code>	<code>function nextPwmOutput( )</code>
<code>node.js</code>	<code>function nextPwmOutput( )</code>
<code>php</code>	<code>function nextPwmOutput( )</code>
<code>cpp</code>	<code>YPwmOutput * nextPwmOutput( )</code>
<code>m</code>	<code>-(YPwmOutput*) nextPwmOutput</code>
<code>pas</code>	<code>function nextPwmOutput( ): TYPwmOutput</code>
<code>vb</code>	<code>function nextPwmOutput( ) As YPwmOutput</code>
<code>cs</code>	<code>YPwmOutput nextPwmOutput( )</code>
<code>java</code>	<code>YPwmOutput nextPwmOutput( )</code>
<code>py</code>	<code>def nextPwmOutput( )</code>

**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.

<b>js</b>	function <b>pulseDurationMove( ms_target, ms_duration)</b>
<b>nodejs</b>	function <b>pulseDurationMove( ms_target, ms_duration)</b>
<b>php</b>	function <b>pulseDurationMove( \$ms_target, \$ms_duration)</b>
<b>cpp</b>	int <b>pulseDurationMove( double ms_target, int ms_duration)</b>
<b>m</b>	- <b>(int) pulseDurationMove : (double) ms_target : (int) ms_duration</b>
<b>pas</b>	function <b>pulseDurationMove( ms_target: double,</b> <b>                          ms_duration: LongInt): LongInt</b>
<b>vb</b>	function <b>pulseDurationMove( ) As Integer</b>
<b>cs</b>	int <b>pulseDurationMove( double ms_target, int ms_duration)</b>
<b>java</b>	int <b>pulseDurationMove( double ms_target, int ms_duration)</b>
<b>py</b>	def <b>pulseDurationMove( ms_target, ms_duration)</b>
<b>cmd</b>	<b>YPwmOutput 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()**  
**pwmoutput→registerValueCallback( )****YPwmOutput**

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

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YPwmOutputValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YPwmOutputValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYPwmOutputValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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 cycle.

<b>js</b>	function <b>set_dutyCycle( newval)</b>
<b>nodejs</b>	function <b>set_dutyCycle( newval)</b>
<b>php</b>	function <b>set_dutyCycle( \$newval)</b>
<b>cpp</b>	int <b>set_dutyCycle( double newval)</b>
<b>m</b>	-(int) setDutyCycle : (double) <b>newval</b>
<b>pas</b>	function <b>set_dutyCycle( newval: double): integer</b>
<b>vb</b>	function <b>set_dutyCycle( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_dutyCycle( double newval)</b>
<b>java</b>	int <b>set_dutyCycle( double newval)</b>
<b>py</b>	def <b>set_dutyCycle( newval)</b>
<b>cmd</b>	YPwmOutput target <b>set_dutyCycle 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 set_dutyCycleAtPowerOn( newval)
nodejs function set_dutyCycleAtPowerOn( newval)
php function set_dutyCycleAtPowerOn( $newval)
cpp int set_dutyCycleAtPowerOn( double newval)
m -(int) setDutyCycleAtPowerOn : (double) newval
pas function set_dutyCycleAtPowerOn( newval: double): integer
vb function set_dutyCycleAtPowerOn( ByVal newval As Double) As Integer
cs int set_dutyCycleAtPowerOn( double newval)
java int set_dutyCycleAtPowerOn( double newval)
py def set_dutyCycleAtPowerOn( newval)
cmd YPwmOutput target set_dutyCycleAtPowerOn newval
```

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()**  
**pwmoutput→setEnabled()****pwmoutput→set\_enabled()**

**YPwmOutput**

Stops or starts the PWM.

<b>js</b>	function <b>set_enabled( newval)</b>
<b>nodejs</b>	function <b>set_enabled( newval)</b>
<b>php</b>	function <b>set_enabled( \$newval)</b>
<b>cpp</b>	int <b>set_enabled( Y_ENABLED_enum newval)</b>
<b>m</b>	-(int) <b>setEnabled : (Y_ENABLED_enum) newval</b>
<b>pas</b>	function <b>set_enabled( newval: Integer): integer</b>
<b>vb</b>	function <b>set_enabled( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_enabled( int newval)</b>
<b>java</b>	int <b>set_enabled( int newval)</b>
<b>py</b>	def <b>set_enabled( newval)</b>
<b>cmd</b>	<b>YPwmOutput 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.

<b>js</b>	function <b>set_enabledAtPowerOn( newval)</b>
<b>nodejs</b>	function <b>set_enabledAtPowerOn( newval)</b>
<b>php</b>	function <b>set_enabledAtPowerOn( \$newval)</b>
<b>cpp</b>	int <b>set_enabledAtPowerOn( Y_ENABLEDATPOWERON_enum newval)</b>
<b>m</b>	-(int) <b>setEnabledAtPowerOn : (Y_ENABLEDATPOWERON_enum) newval</b>
<b>pas</b>	function <b>set_enabledAtPowerOn( newval: Integer): integer</b>
<b>vb</b>	function <b>set_enabledAtPowerOn( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_enabledAtPowerOn( int newval)</b>
<b>java</b>	int <b>set_enabledAtPowerOn( int newval)</b>
<b>py</b>	def <b>set_enabledAtPowerOn( newval)</b>
<b>cmd</b>	<b>YPwmOutput target set_enabledAtPowerOn 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.

<b>js</b>	function <b>set_frequency( newval)</b>
<b>node.js</b>	function <b>set_frequency( newval)</b>
<b>php</b>	function <b>set_frequency( \$newval)</b>
<b>cpp</b>	int <b>set_frequency( int newval)</b>
<b>m</b>	-(int) <b>setFrequency : (int) newval</b>
<b>pas</b>	function <b>set_frequency( newval: LongInt): integer</b>
<b>vb</b>	function <b>set_frequency( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_frequency( int newval)</b>
<b>java</b>	int <b>set_frequency( int newval)</b>
<b>py</b>	def <b>set_frequency( newval)</b>
<b>cmd</b>	YPwmOutput <b>target set_frequency 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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>nodejs</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YPwmOutput 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 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( newval)</b>
nodejs	function <b>set_period( newval)</b>
php	function <b>set_period( \$newval)</b>
cpp	int <b>set_period( double newval)</b>
m	-(int) <b>setPeriod : (double) newval</b>
pas	function <b>set_period( newval: double): integer</b>
vb	function <b>set_period( ByVal newval As Double) As Integer</b>
cs	int <b>set_period( double newval)</b>
java	int <b>set_period( double newval)</b>
py	def <b>set_period( newval)</b>
cmd	<b>YPwmOutput 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()**  
**pwmoutput→setPulseDuration()****pwmoutput→set\_pulseDuration( )**

**YPwmOutput**

Configures the PWM pulses length.

<b>js</b>	function <b>set_pulseDuration( newval)</b>
<b>nodejs</b>	function <b>set_pulseDuration( newval)</b>
<b>php</b>	function <b>set_pulseDuration( \$newval)</b>
<b>cpp</b>	int <b>set_pulseDuration( double newval)</b>
<b>m</b>	-(int) setPulseDuration : (double) <b>newval</b>
<b>pas</b>	function <b>set_pulseDuration( newval: double): integer</b>
<b>vb</b>	function <b>set_pulseDuration( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_pulseDuration( double newval)</b>
<b>java</b>	int <b>set_pulseDuration( double newval)</b>
<b>py</b>	def <b>set_pulseDuration( newval)</b>
<b>cmd</b>	<b>YPwmOutput target set_pulseDuration 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)**  
**pwmoutput→setUserData()****pwmoutput→**  
**set(userData)**

**YPwmOutput**

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

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	- <b>(void) set(userData : (void*) data)</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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');
cpp #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).

### 3. Reference

#### **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)**

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()yFindPwmPowerSource( )****YPwmPowerSource**

Retrieves a voltage source for a given identifier.

<b>js</b>	function <b>yFindPwmPowerSource( func)</b>
<b>node.js</b>	function <b>FindPwmPowerSource( func)</b>
<b>php</b>	function <b>yFindPwmPowerSource( \$func)</b>
<b>cpp</b>	YPwmPowerSource* <b>yFindPwmPowerSource( const string&amp; func)</b>
<b>m</b>	YPwmPowerSource* <b>yFindPwmPowerSource( NSString* func)</b>
<b>pas</b>	function <b>yFindPwmPowerSource( func: string): TYPwmPowerSource</b>
<b>vb</b>	function <b>yFindPwmPowerSource( ByVal func As String) As YPwmPowerSource</b>
<b>cs</b>	YPwmPowerSource <b>FindPwmPowerSource( string func)</b>
<b>java</b>	YPwmPowerSource <b>FindPwmPowerSource( String func)</b>
<b>py</b>	def <b>FindPwmPowerSource( 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()yFirstPwmPowerSource( )

YPwmPowerSource

Starts the enumeration of Voltage sources currently accessible.

```
js function yFirstPwmPowerSource( )
node.js function FirstPwmPowerSource( )
php function yFirstPwmPowerSource( )
cpp YPwmPowerSource* yFirstPwmPowerSource( )
m YPwmPowerSource* yFirstPwmPowerSource( )
pas function yFirstPwmPowerSource( ): TYPwmPowerSource
vb function yFirstPwmPowerSource( ) As YPwmPowerSource
cs YPwmPowerSource FirstPwmPowerSource( )
java YPwmPowerSource FirstPwmPowerSource( )
py def FirstPwmPowerSource( )
```

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>
node.js	function <b>describe( )</b>
php	function <b>describe( )</b>
cpp	string <b>describe( )</b>
m	- <b>(NSString*) 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)=RELAYL01-123456.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 voltage source (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pwmpowersource→get\_advertisedValue()**  
**pwmpowersource→advertisedValue()**  
**pwmpowersource→get\_advertisedValue( )**

**YPwmPowerSource**

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

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

**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→getErrorMessage()**

**YPwmPowerSource**

**pwmPowerSource→errorMessage()**  
**pwmPowerSource**  
**→getErrorMessage( )**

---

Returns the error message of the latest error with the voltage source.

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

**js** function **get\_errorType( )**  
**nodejs** function **get\_errorType( )**  
**php** function **get\_errorType( )**  
**cpp** YRETCODE **get\_errorType( )**  
**pas** function **get\_errorType( )**: YRETCODE  
**vb** function **get\_errorType( )** As YRETCODE  
**cs** YRETCODE **get\_errorType( )**  
**java** int **get\_errorType( )**  
**py** def **get\_errorType( )**

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

**Returns :**

a number corresponding to the code of the latest error that occurred while using the voltage source object

**pwmpowersource→get\_friendlyName()**

**YPwmPowerSource**

**pwmpowersource→friendlyName()**  
**pwmpowersource**  
**→get\_friendlyName( )**

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

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

The returned string uses the logical names of the module and of the voltage source if they are defined, otherwise the serial number of the module and the hardware identifier of the voltage source (for exemple: 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) <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	<b>def 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()`

`YPwmPowerSource`

`pwmpowersource→functionId()pwmpowersource→`  
`get_functionId()`

Returns the hardware identifier of the voltage source, without reference to the module.

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

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()`  
`pwmpowersource→hardwareId()`  
`pwmpowersource→get_hardwareId( )`

**YPwmPowerSource**

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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the voltage source (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**pwmpowersource→get\_logicalName()**  
**pwmpowersource→logicalName()****pwmpowersource**  
**→get\_logicalName( )**

**YPwmPowerSource**

Returns the logical name of the voltage source.

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

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

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

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

**Returns :**

nothing : the result is provided to the callback.

**pwmpowersource→get\_powerMode()**  
**pwmpowersource→powerMode()** **pwmpowersource→get\_powerMode( )**

**YPwmPowerSource**

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)**

**YPwmPowerSource**

**pwmpowersource→userData()pwmpowersource→  
get(userData())**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	function <b>get(userData)</b>
node.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.

**pwmpowersource→isOnline()**  
**pwmpowersource→  
isOnline()****YPwmPowerSource**

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

js	function <b>isOnline( )</b>
node.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 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()****YPwmPowerSource**

Preloads the voltage source cache with a specified validity duration.

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

## 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( ).`

<code>js</code>	<code>function nextPwmPowerSource( )</code>
<code>node.js</code>	<code>function nextPwmPowerSource( )</code>
<code>php</code>	<code>function nextPwmPowerSource( )</code>
<code>cpp</code>	<code>YPwmPowerSource * nextPwmPowerSource( )</code>
<code>m</code>	<code>-(YPwmPowerSource*) nextPwmPowerSource</code>
<code>pas</code>	<code>function nextPwmPowerSource( ): TYPwmPowerSource</code>
<code>vb</code>	<code>function nextPwmPowerSource( ) As YPwmPowerSource</code>
<code>cs</code>	<code>YPwmPowerSource nextPwmPowerSource( )</code>
<code>java</code>	<code>YPwmPowerSource nextPwmPowerSource( )</code>
<code>py</code>	<code>def nextPwmPowerSource( )</code>

**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()****YPwmPowerSource****pwmpowersource→registerValueCallback( )**

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

<b>js</b>	<code>function registerValueCallback( callback)</code>
<b>node.js</b>	<code>function registerValueCallback( callback)</code>
<b>php</b>	<code>function registerValueCallback( \$callback)</code>
<b>cpp</b>	<code>int registerValueCallback( YPwmPowerSourceValueCallback callback)</code>
<b>m</b>	<code>-(int) registerValueCallback : (YPwmPowerSourceValueCallback) callback</code>
<b>pas</b>	<code>function registerValueCallback( callback: TYPwmPowerSourceValueCallback): LongInt</code>
<b>vb</b>	<code>function registerValueCallback( ) As Integer</code>
<b>cs</b>	<code>int registerValueCallback( ValueCallback callback)</code>
<b>java</b>	<code>int registerValueCallback( UpdateCallback callback)</code>
<b>py</b>	<code>def registerValueCallback( callback)</code>

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the 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( newval)</b>
node.js	function <b>set_powerMode( newval)</b>
php	function <b>set_powerMode( \$newval)</b>
cpp	int <b>set_powerMode( Y_POWERMODE_enum newval)</b>
m	-(int) <b>setPowerMode : (Y_POWERMODE_enum) newval</b>
pas	function <b>set_powerMode( newval: Integer): integer</b>
vb	function <b>set_powerMode( ByVal newval As Integer) As Integer</b>
cs	int <b>set_powerMode( int newval)</b>
java	int <b>set_powerMode( int newval)</b>
py	def <b>set_powerMode( newval)</b>
cmd	YPwmPowerSource <b>target set_powerMode 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)****YPwmPowerSource****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>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b> void* <b>data</b>
m	-(void) <b>setUserData</b> : (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	<script type='text/javascript' src='yocto_gyro.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YGyro = yoctolib.YGyro;
php	require_once('yocto_gyro.php');
cpp	#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

#### 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.
<b>qt→get_highestValue()</b>
Returns the maximal value observed for the value since the device was started.
<b>qt→get_logFrequency()</b>
Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>qt→get_logicalName()</b>
Returns the logical name of the quaternion component.
<b>qt→get_lowestValue()</b>
Returns the minimal value observed for the value since the device was started.
<b>qt→get_module()</b>
Gets the YModule object for the device on which the function is located.
<b>qt→get_module_async(callback, context)</b>
Gets the YModule object for the device on which the function is located (asynchronous version).
<b>qt→get_recordedData(startTime, endTime)</b>
Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>qt→get_reportFrequency()</b>
Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>qt→get_resolution()</b>
Returns the resolution of the measured values.
<b>qt→get_unit()</b>
Returns the measuring unit for the value.
<b>qt→get_userData()</b>
Returns the value of the userData attribute, as previously stored using method set(userData).
<b>qt→isOnline()</b>
Checks if the quaternion component is currently reachable, without raising any error.
<b>qt→isOnline_async(callback, context)</b>
Checks if the quaternion component is currently reachable, without raising any error (asynchronous version).
<b>qt→load(msValidity)</b>
Preloads the quaternion component cache with a specified validity duration.
<b>qt→loadCalibrationPoints(rawValues, refValues)</b>
Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>qt→load_async(msValidity, callback, context)</b>
Preloads the quaternion component cache with a specified validity duration (asynchronous version).
<b>qt→nextQt()</b>
Continues the enumeration of quaternion components started using yFirstQt( ).
<b>qt→registerTimedReportCallback(callback)</b>
Registers the callback function that is invoked on every periodic timed notification.
<b>qt→registerValueCallback(callback)</b>
Registers the callback function that is invoked on every change of advertised value.
<b>qt→set_highestValue(newval)</b>
Changes the recorded maximal value observed.
<b>qt→set_logFrequency(newval)</b>
Changes the datalogger recording frequency for this function.
<b>qt→set_logicalName(newval)</b>

### 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()****yFindQt()yFindQt( )****YQt**

Retrieves a quaternion component for a given identifier.

<code>js</code>	<code>function yFindQt( func)</code>
<code>node.js</code>	<code>function FindQt( func)</code>
<code>php</code>	<code>function yFindQt( \$func)</code>
<code>cpp</code>	<code>YQt* yFindQt( string func)</code>
<code>m</code>	<code>+ (YQt*) yFindQt : (NSString*) func</code>
<code>pas</code>	<code>function yFindQt( func: string): TYQt</code>
<code>vb</code>	<code>function yFindQt( ByVal func As String) As YQt</code>
<code>cs</code>	<code>YQt FindQt( string func)</code>
<code>java</code>	<code>YQt FindQt( String func)</code>
<code>py</code>	<code>def FindQt( 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 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()yFirstQt( )

YQt

Starts the enumeration of quaternion components currently accessible.

js	function <b>yFirstQt()</b>
node.js	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)=RELAYL01-123456.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 quaternion component (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.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 get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YSensor target get_advertisedValue
```

**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()**  
**qt→currentRawValue()qt→**  
**get\_currentRawValue( )****YQt**

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

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( )**: double  
**vb** function **get\_currentRawValue( )** As Double  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YSensor **target get\_currentRawValue**

**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( ): double</b>
vb	function <b>get_currentValue( ) As Double</b>
cs	double <b>get_currentValue( )</b>
java	double <b>get_currentValue( )</b>
py	def <b>get_currentValue( )</b>
cmd	YSensor target <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()**  
**qt→errorMessage()qt→get\_errorMessage( )**

YQt

Returns the error message of the latest error with the quaternion component.

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

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

**Returns :**

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

## qt->get\_errorType() qt->errorType()qt->get\_errorType( )

YQt

Returns the numerical error code of the latest error with the quaternion component.

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

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

**Returns :**

a number corresponding to the code of the latest error that 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 get_friendlyName( )
nodejs function get_friendlyName( )
php function get_friendlyName( )
cpp string get_friendlyName( )
m -(NSString*) friendlyName
cs string get_friendlyName( )
java String get_friendlyName( )
py def get_friendlyName( )
```

The returned string uses the logical names of the module and of the quaternion component if they are defined, otherwise the serial number of the module and the hardware identifier of the quaternion component (for exemple: 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()  
qt→functionDescriptor()qt→  
get\_functionDescriptor( )****YQt**

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

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR 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	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def 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()  
qt→functionId()qt→get\_functionId()****YQt**

Returns the hardware identifier of the quaternion component, without reference to the module.

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

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the quaternion component. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the quaternion component (ex: RELAYL01-123456.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 get_highestValue( )
node.js function get_highestValue( )
php function get_highestValue( )
cpp double get_highestValue( )
m -(double) highestValue
pas function get_highestValue( ): double
vb function get_highestValue( ) As Double
cs double get_highestValue( )
java double get_highestValue( )
py def get_highestValue( )
cmd YSensor target get_highestValue
```

**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( ): string</b>
vb	function <b>get_logFrequency( ) As String</b>
cs	string <b>get_logFrequency( )</b>
java	String <b>get_logFrequency( )</b>
py	def <b>get_logFrequency( )</b>
cmd	YSensor <b>target 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()**  
**qt→logicalName()qt→get\_logicalName( )****YQt**

Returns the logical name of the quaternion component.

**js** function **get\_logicalName( )****node.js** function **get\_logicalName( )****php** function **get\_logicalName( )****cpp** string **get\_logicalName( )****m** -(NSString\*) logicalName**pas** function **get\_logicalName( )**: string**vb** function **get\_logicalName( )** As String**cs** string **get\_logicalName( )****java** String **get\_logicalName( )****py** def **get\_logicalName( )****cmd** YSensor target **get\_logicalName****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()**

YQt

**qt→lowestValue()qt→get\_lowestValue()**

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

```
js   function get_lowestValue( )  
nodejs function get_lowestValue( )  
php  function get_lowestValue( )  
cpp   double get_lowestValue( )  
m    -(double) lowestValue  
pas   function get_lowestValue( ): double  
vb    function get_lowestValue( ) As Double  
cs    double get_lowestValue( )  
java  double get_lowestValue( )  
py    def get_lowestValue( )  
cmd   YSensor target get_lowestValue
```

**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()**  
**qt→module()qt→get\_module( )****YQt**

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

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

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

**Returns :**

an instance of `YModule`

## qt→get\_module\_async() qt→module\_async()

YQt

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

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

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

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

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

### Returns :

nothing : the result is provided to the callback.

## **qt→get\_recordedData()**

**YQt**

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

js	function <b>get_recordedData( startTime, endTime)</b>
node.js	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	YDataSet <b>get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime : (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
java	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
py	<b>def get_recordedData( startTime, endTime)</b>
cmd	YSensor <b>target get_recordedData startTime 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 get_reportFrequency( )
nodejs function get_reportFrequency( )
php   function get_reportFrequency( )
cpp   string get_reportFrequency( )
m     -(NSString*) reportFrequency
pas   function get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs    string get_reportFrequency( )
java  String get_reportFrequency( )
py    def get_reportFrequency( )
cmd   YSensor target get_reportFrequency
```

**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( )**

YQt

Returns the resolution of the measured values.

```
js function get_resolution( )
node.js function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YSensor target get_resolution
```

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() qt→unit()qt→get\_unit( )

YQt

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*) <b>unit</b>
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 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**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	<code>function get(userData) {</code>
node.js	<code>function get(userData) {</code>
php	<code>function get(userData) {</code>
cpp	<code>void * get(userData) {</code>
m	<code>-(void*) userData</code>
pas	<code>function get(userData): Tobject {</code>
vb	<code>function get(userData) As Object {</code>
cs	<code>object get(userData) {</code>
java	<code>Object get(userData) {</code>
py	<code>def get(userData):</code>

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()****YQt**

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

js	function <b>isOnline( )</b>
node.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 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()****YQt**

Preloads the quaternion component cache with a specified validity duration.

js	function <b>load( msValidity)</b>
node.js	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-( <b>YRETCODE</b> ) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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 **nextQt( )****nodejs** function **nextQt( )****php** function **nextQt( )****cpp** **YQt \* nextQt( )****m** -(**YQt\***) **nextQt****pas** function **nextQt( )**: TYQt**vb** function **nextQt( )** As YQt**cs** **YQt nextQt( )****java** **YQt nextQt( )****py** def **nextQt( )****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 registerTimedReportCallback( callback)
nodejs function registerTimedReportCallback( callback)
php  function registerTimedReportCallback( $callback)
cpp   int registerTimedReportCallback( YQtTimedReportCallback callback)
m    -(int) registerTimedReportCallback : (YQtTimedReportCallback) callback
pas   function registerTimedReportCallback( callback: TYQtTimedReportCallback): LongInt
vb    function registerTimedReportCallback( ) As Integer
cs    int registerTimedReportCallback( TimedReportCallback callback)
java  int registerTimedReportCallback( TimedReportCallback callback)
py    def registerTimedReportCallback( callback)
```

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

### Parameters :

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and 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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YQtValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YQtValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYQtValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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 set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YSensor target set_highestValue newval
```

**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() qt→setLogFrequency()qt→set\_logFrequency( )

YQt

Changes the datalogger recording frequency for this function.

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

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

**Parameters :**

**newval** a string corresponding to the logical name of the quaternion component.

**Returns :**

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

**qt→set\_lowestValue()**  
**qt→setLowestValue()qt→set\_lowestValue( )**

YQt

Changes the recorded minimal value observed.

```
js function set_lowestValue( newval)
node.js 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 YSensor target set_lowestValue newval
```

**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()**  
**qt→setReportFrequency()qt→**  
**set\_reportFrequency( )**

YQt

Changes the timed value notification frequency for this function.

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

## qt→set\_resolution() qt→setResolution()qt→set\_resolution()

YQt

Changes the resolution of the measured physical values.

```
js function set_resolution( newval)
node.js function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YSensor target set_resolution newval
```

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)**  
**qt→setUserData()qt→set(userData) ( )**

YQt

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

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

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

**Parameters :**

**data** any kind of object to be stored

## 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_realtimedclock.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YRealTimeClock = yoctolib.YRealTimeClock;
php	require_once('yocto_realtimedclock.php');
cpp	#include "yocto_realtimedclock.h"
m	#import "yocto_realtimedclock.h"
pas	uses yocto_realtimedclock;
vb	yocto_realtimedclock.vb
cs	yocto_realtimedclock.cs
java	import com.yoctopuce.YoctoAPI.YRealTimeClock;
py	from yocto_realtimedclock 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()

### 3. Reference

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

#### **realtimeclock→get\_module\_async(callback, context)**

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

#### **realtimeclock→get\_timeSet()**

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

#### **realtimeclock→get\_unixTime()**

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

#### **realtimeclock→get\_userData()**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

#### **realtimeclock→get\_utcOffset()**

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

#### **realtimeclock→isOnline()**

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

#### **realtimeclock→isOnline\_async(callback, context)**

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

#### **realtimeclock→load(msValidity)**

Preloads the clock cache with a specified validity duration.

#### **realtimeclock→load\_async(msValidity, callback, context)**

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

#### **realtimeclock→nextRealTimeClock()**

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

#### **realtimeclock→registerValueCallback(callback)**

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

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

Changes the logical name of the clock.

#### **realtimeclock→set\_unixTime(newval)**

Changes the current time.

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

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

#### **realtimeclock→set\_utcOffset(newval)**

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

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

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

**YRealTimeClock.FindRealTimeClock()****YRealTimeClock****yFindRealTimeClock()yFindRealTimeClock( )**

Retrieves a clock for a given identifier.

<code>js</code>	<code>function yFindRealTimeClock( func)</code>
<code>nodejs</code>	<code>function FindRealTimeClock( func)</code>
<code>php</code>	<code>function yFindRealTimeClock( \$func)</code>
<code>cpp</code>	<code>YRealTimeClock* yFindRealTimeClock( const string&amp; func)</code>
<code>m</code>	<code>YRealTimeClock* yFindRealTimeClock( NSString* func)</code>
<code>pas</code>	<code>function yFindRealTimeClock( func: string): TYRealTimeClock</code>
<code>vb</code>	<code>function yFindRealTimeClock( ByVal func As String) As YRealTimeClock</code>
<code>cs</code>	<code>YRealTimeClock FindRealTimeClock( string func)</code>
<code>java</code>	<code>YRealTimeClock FindRealTimeClock( String func)</code>
<code>py</code>	<code>def FindRealTimeClock( 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 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()****YRealTimeClock****yFirstRealTimeClock()yFirstRealTimeClock( )**

Starts the enumeration of clocks currently accessible.

js	function <b>yFirstRealTimeClock( )</b>
node.js	function <b>FirstRealTimeClock( )</b>
php	function <b>yFirstRealTimeClock( )</b>
cpp	YRealTimeClock* <b>yFirstRealTimeClock( )</b>
m	YRealTimeClock* <b>yFirstRealTimeClock( )</b>
pas	function <b>yFirstRealTimeClock( ): TYRealTimeClock</b>
vb	function <b>yFirstRealTimeClock( ) As YRealTimeClock</b>
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	- <b>(NSString*) 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)=RELAYL01-123456.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 clock (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**realtimeclock→get\_advertisedValue()**  
**realtimeclock→advertisedValue()realtimeclock→**  
**get\_advertisedValue( )**

**YRealTimeClock**

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

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

**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	<b>def 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→getErrorMessage()**  
**realtimeclock→errorMessage()** **realtimeclock→getErrorMessage( )**

**YRealTimeClock**

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

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

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.

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

**realtimeclock→get\_friendlyName()** **YRealTimeClock**  
**realtimeclock→friendlyName()realtimeclock→**  
**get\_friendlyName( )**

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

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

The returned string uses the logical names of the module and of the 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.

---

<b>realtimeclock→get_functionDescriptor()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→functionDescriptor()realtimeclock →get_functionDescriptor( )</b>	

---

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.

**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()**  
**realtimeclock→hardwareId()** **realtimeclock→get\_hardwareId( )**

**YRealTimeClock**

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

js	function <b>get_hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the clock (ex: RELAYL01-123456.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 **get\_logicalName( )**  
**nodejs** function **get\_logicalName( )**  
**php** function **get\_logicalName( )**  
**cpp** string **get\_logicalName( )**  
**m** -(NSString\*) logicalName  
**pas** function **get\_logicalName( )**: string  
**vb** function **get\_logicalName( )** As String  
**cs** string **get\_logicalName( )**  
**java** String **get\_logicalName( )**  
**py** def **get\_logicalName( )**  
**cmd** YRealTimeClock **target get\_logicalName**

**Returns :**

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

**realtimeclock→get\_module()**  
**realtimeclock→module()realtimeclock→**  
**get\_module()**

**YRealTimeClock**

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()**  
**realtimeclock→module\_async()****YRealTimeClock**

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

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**realtimeclock→get\_timeSet()**  
**realtimeclock→timeSet()realtimeclock→**  
**get\_timeSet()**

**YRealTimeClock**

Returns true if the clock has been set, and false otherwise.

```
js function get_timeSet( )  
nodejs function get_timeSet( )  
php function get_timeSet( )  
cpp Y_TIMESET_enum get_timeSet( )  
m -(Y_TIMESET_enum) timeSet  
pas function get_timeSet( ): Integer  
vb function get_timeSet( ) As Integer  
cs int get_timeSet( )  
java int get_timeSet( )  
py def get_timeSet( )  
cmd YRealTimeClock target get_timeSet
```

**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()**  
**realtimeclock→unixTime()** **realtimeclock→get\_unixTime( )**

**YRealTimeClock**

Returns the current time in Unix format (number of elapsed seconds since Jan 1st, 1970).

**js** function **get\_unixTime( )**  
**nodejs** function **get\_unixTime( )**  
**php** function **get\_unixTime( )**  
**cpp** s64 **get\_unixTime( )**  
**m** -(s64) **unixTime**  
**pas** function **get\_unixTime( )**: int64  
**vb** function **get\_unixTime( )** As Long  
**cs** long **get\_unixTime( )**  
**java** long **get\_unixTime( )**  
**py** def **get\_unixTime( )**  
**cmd** YRealTimeClock **target get\_unixTime**

**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)**  
**realtimeclock→userData()** **realtimeclock→get(userData)**

**YRealTimeClock**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	function <b>get(userData)</b>
node.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.

**realtimeclock→get\_utcOffset()**  
**realtimeclock→utcOffset()** **realtimeclock→get\_utcOffset( )**

**YRealTimeClock**

Returns the number of seconds between current time and UTC time (time zone).

**js** function **get\_utcOffset( )**  
**nodejs** function **get\_utcOffset( )**  
**php** function **get\_utcOffset( )**  
**cpp** int **get\_utcOffset( )**  
**m** -(int) **utcOffset**  
**pas** function **get\_utcOffset( )**: LongInt  
**vb** function **get\_utcOffset( )** As Integer  
**cs** int **get\_utcOffset( )**  
**java** int **get\_utcOffset( )**  
**py** def **get\_utcOffset( )**  
**cmd** YRealTimeClock **target get\_utcOffset**

**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()****YRealTimeClock**

Preloads the clock 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.

**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	<b>YRealTimeClock * nextRealTimeClock( )</b>
m	<b>-(YRealTimeClock*) nextRealTimeClock</b>
pas	function <b>nextRealTimeClock( ): TYRealTimeClock</b>
vb	function <b>nextRealTimeClock( ) As YRealTimeClock</b>
cs	<b>YRealTimeClock nextRealTimeClock( )</b>
java	<b>YRealTimeClock nextRealTimeClock( )</b>
py	<b>def 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()****YRealTimeClock****realtimeclock→registerValueCallback( )**

Registers the callback function that is invoked on every change of advertised value.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YRealTimeClockValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YRealTimeClockValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYRealTimeClockValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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.

<b>realtimeclock→set_logicalName()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→setLogicalName()realtimeclock→set_logicalName( )</b>	

Changes the logical name of the clock.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>nodejs</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YRealTimeClock 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()** **YRealTimeClock**  
**realtimeclock→setUnixTime()realtimeclock→**  
**set\_unixTime( )**

Changes the current time.

```
js function set_unixTime( newval)
nodejs function set_unixTime( newval)
php function set_unixTime( $newval)
cpp int set_unixTime( s64 newval)
m -(int) setUnixTime : (s64) newval
pas function set_unixTime( newval: int64): integer
vb function set_unixTime( ByVal newval As Long) As Integer
cs int set_unixTime( long newval)
java int set_unixTime( long newval)
py def set_unixTime( newval)
cmd YRealTimeClock target set_unixTime newval
```

Time is specified 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)****YRealTimeClock****realtimeclock→setUserData()realtimeclock→  
set(userData)**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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()** **YRealTimeClock**  
**realtimeclock→setUtcOffset()** **realtimeclock→set\_utcOffset( )**

Changes the number of seconds between current time and UTC time (time zone).

```
js function set_utcOffset( newval)
nodejs function set_utcOffset( newval)
php function set_utcOffset( $newval)
cpp int set_utcOffset( int newval)
m -(int) setUtcOffset : (int) newval
pas function set_utcOffset( newval: LongInt): integer
vb function set_utcOffset( ByVal newval As Integer) As Integer
cs int set_utcOffset( int newval)
java int set_utcOffset( int newval)
py def set_utcOffset( newval)
cmd YRealTimeClock target set_utcOffset newval
```

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 <script type='text/javascript' src='yocto_refframe.js'></script>
nodejs var yoctolib = require('yoctolib');
var YRefFrame = yoctolib.YRefFrame;
php require_once('yocto_refframe.php');
cpp #include "yocto_refframe.h"
m #import "yocto_refframe.h"
pas uses yocto_refframe;
vb yocto_refframe.vb
cs yocto_refframe.cs
java import com.yoctopuce.YoctoAPI.YRefFrame;
py from yocto_refframe import *

```

### 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()****YRefFrame****yFindRefFrame()yFindRefFrame( )**

Retrieves a reference frame for a given identifier.

<b>js</b>	<code>function yFindRefFrame( func)</code>
<b>nodejs</b>	<code>function FindRefFrame( func)</code>
<b>php</b>	<code>function yFindRefFrame( \$func)</code>
<b>cpp</b>	<code>YRefFrame* yFindRefFrame( const string&amp; func)</code>
<b>m</b>	<code>YRefFrame* yFindRefFrame( NSString* func)</code>
<b>pas</b>	<code>function yFindRefFrame( func: string): TYRefFrame</code>
<b>vb</b>	<code>function yFindRefFrame( ByVal func As String) As YRefFrame</code>
<b>cs</b>	<code>YRefFrame FindRefFrame( string func)</code>
<b>java</b>	<code>YRefFrame FindRefFrame( String func)</code>
<b>py</b>	<code>def FindRefFrame( 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 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()yFirstRefFrame( )**

Starts the enumeration of reference frames currently accessible.

js	function <b>yFirstRefFrame( )</b>
node.js	function <b>FirstRefFrame( )</b>
php	function <b>yFirstRefFrame( )</b>
cpp	<b>YRefFrame*</b> <b>yFirstRefFrame( )</b>
m	<b>YRefFrame*</b> <b>yFirstRefFrame( )</b>
pas	function <b>yFirstRefFrame( ): TYRefFrame</b>
vb	function <b>yFirstRefFrame( ) As YRefFrame</b>
cs	<b>YRefFrame FirstRefFrame( )</b>
java	<b>YRefFrame 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>
node.js	function <b>cancel3DCalibration( )</b>
php	function <b>cancel3DCalibration( )</b>
cpp	int <b>cancel3DCalibration( )</b>
m	-(int) <b>cancel3DCalibration</b>
pas	function <b>cancel3DCalibration( ): LongInt</b>
vb	function <b>cancel3DCalibration( ) As Integer</b>
cs	int <b>cancel3DCalibration( )</b>
java	int <b>cancel3DCalibration( )</b>
py	def <b>cancel3DCalibration( )</b>
cmd	<b>YRefFrame target cancel3DCalibration</b>

On failure, throws an exception or returns a negative error code.

**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 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**refframe→get\_3DCalibrationHint()**  
**refframe→3DCalibrationHint()****refframe→get\_3DCalibrationHint()**

**YRefFrame**

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*) <b>3DCalibrationHint</b>
pas	function <b>get_3DCalibrationHint( ): string</b>
vb	function <b>get_3DCalibrationHint( ) As String</b>
cs	string <b>get_3DCalibrationHint( )</b>
java	<b>String get_3DCalibrationHint( )</b>
py	<b>def get_3DCalibrationHint( )</b>
cmd	<b>YRefFrame target get_3DCalibrationHint</b>

**Returns :**  
a character string.

`refframe→get_3DCalibrationLogMsg()`  
`refframe→3DCalibrationLogMsg()``refframe→get_3DCalibrationLogMsg( )`

**YRefFrame**

Returns the latest log message from the calibration process.

```
js function get_3DCalibrationLogMsg( )
nodejs function get_3DCalibrationLogMsg( )
php function get_3DCalibrationLogMsg( )
cpp string get_3DCalibrationLogMsg( )
m -(NSString*) 3DCalibrationLogMsg
pas function get_3DCalibrationLogMsg( ): string
vb function get_3DCalibrationLogMsg( ) As String
cs string get_3DCalibrationLogMsg( )
java String get_3DCalibrationLogMsg( )
py def get_3DCalibrationLogMsg( )
cmd YRefFrame target get_3DCalibrationLogMsg
```

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.

<b>js</b>	function <b>get_3DCalibrationProgress( )</b>
<b>node.js</b>	function <b>get_3DCalibrationProgress( )</b>
<b>php</b>	function <b>get_3DCalibrationProgress( )</b>
<b>cpp</b>	<b>int get_3DCalibrationProgress( )</b>
<b>m</b>	<b>-(int) 3DCalibrationProgress</b>
<b>pas</b>	function <b>get_3DCalibrationProgress( ): LongInt</b>
<b>vb</b>	function <b>get_3DCalibrationProgress( ) As Integer</b>
<b>cs</b>	<b>int get_3DCalibrationProgress( )</b>
<b>java</b>	<b>int get_3DCalibrationProgress( )</b>
<b>py</b>	<b>def get_3DCalibrationProgress( )</b>
<b>cmd</b>	<b>YRefFrame target get_3DCalibrationProgress</b>

**Returns :**

an integer between 0 (not started) and 100 (stage completed).

`refframe→get_3DCalibrationStage()`  
`refframe→3DCalibrationStage()refframe→`  
`get_3DCalibrationStage( )`

**YRefFrame**

Returns index of the current stage of the calibration initiated with method `start3DCalibration`.

`js` function `get_3DCalibrationStage( )`  
`nodejs` function `get_3DCalibrationStage( )`  
`php` function `get_3DCalibrationStage( )`  
`cpp` int `get_3DCalibrationStage( )`  
`m` -(int) `3DCalibrationStage`  
`pas` function `get_3DCalibrationStage( )`: LongInt  
`vb` function `get_3DCalibrationStage( )` As Integer  
`cs` int `get_3DCalibrationStage( )`  
`java` int `get_3DCalibrationStage( )`  
`py` def `get_3DCalibrationStage( )`  
`cmd` YRefFrame target `get_3DCalibrationStage`

**Returns :**

an integer, growing each time a calibration stage is completed.

**refframe→get\_3DCalibrationStageProgress()**  
**refframe→3DCalibrationStageProgress()refframe→**  
**get\_3DCalibrationStageProgress( )**

**YRefFrame**

Returns the process indicator for the current stage of the calibration initiated with method start3DCalibration.

```
js function get_3DCalibrationStageProgress( )
nodejs function get_3DCalibrationStageProgress( )
php function get_3DCalibrationStageProgress( )
cpp int get_3DCalibrationStageProgress( )
m -(int) 3DCalibrationStageProgress
pas function get_3DCalibrationStageProgress( ): LongInt
vb function get_3DCalibrationStageProgress( ) As Integer
cs int get_3DCalibrationStageProgress( )
java int get_3DCalibrationStageProgress( )
py def get_3DCalibrationStageProgress( )
cmd YRefFrame target get_3DCalibrationStageProgress
```

**Returns :**

an integer between 0 (not started) and 100 (stage completed).

`refframe→get_advertisedValue()`  
`refframe→advertisedValue()refframe→`  
`get_advertisedValue( )`

**YRefFrame**

Returns the current value of the reference frame (no more than 6 characters).

`js` function `get_advertisedValue( )`  
`nodejs` function `get_advertisedValue( )`  
`php` function `get_advertisedValue( )`  
`cpp` string `get_advertisedValue( )`  
`m` -(NSString\*) `advertisedValue`  
`pas` function `get_advertisedValue( ): string`  
`vb` function `get_advertisedValue( ) As String`  
`cs` string `get_advertisedValue( )`  
`java` String `get_advertisedValue( )`  
`py` def `get_advertisedValue( )`  
`cmd` YRefFrame target `get_advertisedValue`

**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( ): double</b>
vb	function <b>get_bearing( ) As Double</b>
cs	double <b>get_bearing( )</b>
java	double <b>get_bearing( )</b>
py	def <b>get_bearing( )</b>
cmd	<b>YRefFrame target 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→getErrorMessage()**  
**refframe→errorMessage()** refframe→  
**getErrorMessage( )**

**YRefFrame**

Returns the error message of the latest error with the reference frame.

js	function getErrorMessage( )
nodejs	function getErrorMessage( )
php	function getErrorMessage( )
cpp	string getErrorMessage( )
m	-(NSString*) errorMessage
pas	function getErrorMessage( ): string
vb	function getErrorMessage( ) As String
cs	string getErrorMessage( )
java	String getErrorMessage( )
py	def getErrorMessage( )

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()`  
`refframe→friendlyName()`  
`refframe→get_friendlyName( )`

**YRefFrame**

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>
cpp	string <b>get_friendlyName( )</b>
m	- <b>(NSString*) friendlyName</b>
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 exemple: 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()**  
**refframe→functionDescriptor()refframe→**  
**get\_functionDescriptor( )**

**YRefFrame**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor( )</b>
node.js	function <b>get_functionDescriptor( )</b>
php	function <b>get_functionDescriptor( )</b>
cpp	<b>YFUN_DESCR 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	<b>YFUN_DESCR get_functionDescriptor( )</b>
java	<b>String get_functionDescriptor( )</b>
py	<b>def 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()**  
**refframe→hardwareId()refframe→**  
**get\_hardwareId( )**

**YRefFrame**

Returns the unique hardware identifier of the reference frame in the form SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.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( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	<b>String get_hardwareId( )</b>
py	<b>def 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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the reference frame (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**refframe→get\_logicalName()**  
**refframe→logicalName()** **refframe→get\_logicalName( )**

**YRefFrame**

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	<b>YRefFrame target 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 online.

**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 get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**refframe→get\_mountOrientation()**  
**refframe→mountOrientation()** **refframe→get\_mountOrientation( )**

**YRefFrame**

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.

<b>js</b>	function <b>get_mountOrientation( )</b>
<b>node.js</b>	function <b>get_mountOrientation( )</b>
<b>php</b>	function <b>get_mountOrientation( )</b>
<b>cpp</b>	<b>Y_MOUNTORIENTATION get_mountOrientation( )</b>
<b>m</b>	-( <b>Y_MOUNTORIENTATION</b> ) <b>mountOrientation</b>
<b>pas</b>	function <b>get_mountOrientation( )</b> : <b>TYMOUNTORIENTATION</b>
<b>vb</b>	function <b>get_mountOrientation( )</b> As <b>Y_MOUNTORIENTATION</b>
<b>cs</b>	<b>MOUNTORIENTATION get_mountOrientation( )</b>
<b>java</b>	<b>MOUNTORIENTATION get_mountOrientation( )</b>
<b>py</b>	def <b>get_mountOrientation( )</b>
<b>cmd</b>	<b>YRefFrame target 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()`  
`refframe→mountPosition()``refframe→get_mountPosition()`

**YRefFrame**

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 get_mountPosition( )
nodejs function get_mountPosition( )
php function get_mountPosition( )
cpp Y_MOUNTPOSITION get_mountPosition( )
m -(Y_MOUNTPOSITION) mountPosition
pas function get_mountPosition( ): TYMOUNTPOSITION
vb function get_mountPosition( ) As Y_MOUNTPOSITION
cs MOUNTPOSITION get_mountPosition( )
java MOUNTPOSITION get_mountPosition( )
py def get_mountPosition( )
cmd YRefFrame target get_mountPosition
```

**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	<code>function get(userData) </code>
nodejs	<code>function get(userData) </code>
php	<code>function get(userData) </code>
cpp	<code>void * get(userData) </code>
m	<code>-(void*) userData</code>
pas	<code>function get(userData): Tobject</code>
vb	<code>function get(userData) As Object</code>
cs	<code>object get(userData) </code>
java	<code>Object get(userData) </code>
py	<code>def get(userData) </code>

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()****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()****YRefFrame**

Preloads the reference frame cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-( <b>YRETCODE</b> ) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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 more3DCalibration( )
nodejs function more3DCalibration( )
php function more3DCalibration( )
cpp int more3DCalibration( )
m -(int) more3DCalibration
pas function more3DCalibration( ): LongInt
vb function more3DCalibration( ) As Integer
cs int more3DCalibration( )
java int more3DCalibration( )
py def more3DCalibration( )
cmd YRefFrame target more3DCalibration
```

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	<b>YRefFrame * nextRefFrame( )</b>
m	<b>-(YRefFrame*) nextRefFrame</b>
pas	function <b>nextRefFrame( )</b> : TYRefFrame
vb	function <b>nextRefFrame( )</b> As YRefFrame
cs	<b>YRefFrame nextRefFrame( )</b>
java	<b>YRefFrame nextRefFrame( )</b>
py	<b>def 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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YRefFrameValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YRefFrameValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYRefFrameValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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 save3DCalibration( )
nodejs function save3DCalibration( )
php function save3DCalibration( )
cpp int save3DCalibration( )
m -(int) save3DCalibration
pas function save3DCalibration( ): LongInt
vb function save3DCalibration( ) As Integer
cs int save3DCalibration( )
java int save3DCalibration( )
py def save3DCalibration( )
cmd YRefFrame target save3DCalibration
```

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 <b>set_bearing( newval)</b>
node.js	function <b>set_bearing( newval)</b>
php	function <b>set_bearing( \$newval)</b>
cpp	int <b>set_bearing( double newval)</b>
m	- (int) <b>setBearing : (double) newval</b>
pas	function <b>set_bearing( newval: double): integer</b>
vb	function <b>set_bearing( ByVal newval As Double) As Integer</b>
cs	int <b>set_bearing( double newval)</b>
java	int <b>set_bearing( double newval)</b>
py	def <b>set_bearing( newval)</b>
cmd	<b>YRefFrame target set_bearing newval</b>

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()**  
**refframe→setLogicalName()** **refframe→**  
**set\_logicalName( )**

**YRefFrame**

Changes the logical name of the reference frame.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YRefFrame 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 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)**  
**refframe→setUserData()** **refframe→**  
**set(userData)**

**YRefFrame**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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 start3DCalibration( )
node.js function start3DCalibration( )
php function start3DCalibration( )
cpp int start3DCalibration( )
m -(int) start3DCalibration
pas function start3DCalibration( ): LongInt
vb function start3DCalibration( ) As Integer
cs int start3DCalibration( )
java int start3DCalibration( )
py def start3DCalibration( )
cmd YRefFrame target start3DCalibration
```

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	<script type='text/javascript' src='yocto_relay.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YRelay = yoctolib.YRelay;
php	require_once('yocto_relay.php');
cpp	#include "yocto_relay.h"
m	#import "yocto_relay.h"
pas	uses yocto_relay;
vb	yocto_relay.vb
cs	yocto_relay.cs
java	import com.yoctopuce.YoctoAPI.YRelay;
py	from yocto_relay import *

### 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()yFindRelay( )**

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()yFirstRelay()**

Starts the enumeration of relays currently accessible.

```
js function yFirstRelay( )
node.js function FirstRelay( )
php function yFirstRelay( )
cpp YRelay* yFirstRelay( )
m YRelay* yFirstRelay( )
pas function yFirstRelay( ): TYRelay
vb function yFirstRelay( ) As YRelay
cs YRelay FirstRelay( )
java YRelay FirstRelay( )
def FirstRelay( )
```

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()****YRelay**

Schedules a pulse.

```
js function delayedPulse( ms_delay, ms_duration)
node.js function delayedPulse( ms_delay, ms_duration)
php function delayedPulse( $ms_delay, $ms_duration)
cpp int delayedPulse( int ms_delay, int ms_duration)
m -(int) delayedPulse : (int) ms_delay : (int) ms_duration
pas function delayedPulse( ms_delay: LongInt, ms_duration: LongInt): integer
vb function delayedPulse( ByVal ms_delay As Integer,
                           ByVal ms_duration As Integer) As Integer
cs int delayedPulse( int ms_delay, int ms_duration)
java int delayedPulse( int ms_delay, int ms_duration)
py def delayedPulse( ms_delay, ms_duration)
cmd YRelay target delayedPulse ms_delay ms_duration
```

**Parameters :**

**ms\_delay** waiting time before the pulse, in millisecondes

**ms\_duration** pulse duration, in millisecondes

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**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 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**relay→get\_advertisedValue()**  
**relay→advertisedValue()relay→**  
**get\_advertisedValue( )**

**YRelay**

Returns the current value of the relay (no more than 6 characters).

<code>js</code>	<code>function get_advertisedValue( )</code>
<code>nodejs</code>	<code>function get_advertisedValue( )</code>
<code>php</code>	<code>function get_advertisedValue( )</code>
<code>cpp</code>	<code>string get_advertisedValue( )</code>
<code>m</code>	<code>-(NSString*) advertisedValue</code>
<code>pas</code>	<code>function get_advertisedValue( ): string</code>
<code>vb</code>	<code>function get_advertisedValue( ) As String</code>
<code>cs</code>	<code>string get_advertisedValue( )</code>
<code>java</code>	<code>String get_advertisedValue( )</code>
<code>py</code>	<code>def get_advertisedValue( )</code>
<code>cmd</code>	<code>YRelay target get_advertisedValue</code>

**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) <b>countdown</b>
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 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

---

<b>relay-&gt;get_errorType()</b>	<b>YRelay</b>
<b>relay-&gt;errorType()relay-&gt;get_errorType( )</b>	

---

Returns the numerical error code of the latest error with the relay.

<code>js</code>	<code>function get_errorType( )</code>
<code>node.js</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 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 get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

The returned string uses the logical names of the module and of the relay if they are defined, otherwise the serial number of the module and the hardware identifier of the relay (for exemple: 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()**  
**relay→functionDescriptor()relay→**  
**get\_functionDescriptor( )**

**YRelay**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

<code>js</code>	function <b>get_functionDescriptor( )</b>
<code>nodejs</code>	function <b>get_functionDescriptor( )</b>
<code>php</code>	function <b>get_functionDescriptor( )</b>
<code>cpp</code>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<code>m</code>	-(YFUN_DESCR) <b>functionDescriptor</b>
<code>pas</code>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<code>vb</code>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<code>cs</code>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<code>java</code>	String <b>get_functionDescriptor( )</b>
<code>py</code>	<b>def 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( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def 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>
node.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( ) As String</b>
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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the relay (ex: RELAYL01-123456.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 get_logicalName( )
nodejs function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YRelay target get_logicalName
```

**Returns :**

a string corresponding to the logical name of the relay. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

`relay->get_maxTimeOnStateA()`  
`relay->maxTimeOnStateA()relay->`  
`get_maxTimeOnStateA( )`

YRelay

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

<code>js</code>	<code>function get_maxTimeOnStateA( )</code>
<code>nodejs</code>	<code>function get_maxTimeOnStateA( )</code>
<code>php</code>	<code>function get_maxTimeOnStateA( )</code>
<code>cpp</code>	<code>s64 get_maxTimeOnStateA( )</code>
<code>m</code>	<code>-(s64) maxTimeOnStateA</code>
<code>pas</code>	<code>function get_maxTimeOnStateA( ): int64</code>
<code>vb</code>	<code>function get_maxTimeOnStateA( ) As Long</code>
<code>cs</code>	<code>long get_maxTimeOnStateA( )</code>
<code>java</code>	<code>long get_maxTimeOnStateA( )</code>
<code>py</code>	<code>def get_maxTimeOnStateA( )</code>
<code>cmd</code>	<code>YRelay target get_maxTimeOnStateA</code>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEA\_INVALID.

**relay→get\_maxTimeOnStateB()**  
**relay→maxTimeOnStateB()relay→**  
**get\_maxTimeOnStateB( )**

**YRelay**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

<b>js</b>	function <b>get_maxTimeOnStateB( )</b>
<b>node.js</b>	function <b>get_maxTimeOnStateB( )</b>
<b>php</b>	function <b>get_maxTimeOnStateB( )</b>
<b>cpp</b>	s64 <b>get_maxTimeOnStateB( )</b>
<b>m</b>	-(s64) maxTimeOnStateB
<b>pas</b>	function <b>get_maxTimeOnStateB( ): int64</b>
<b>vb</b>	function <b>get_maxTimeOnStateB( ) As Long</b>
<b>cs</b>	long <b>get_maxTimeOnStateB( )</b>
<b>java</b>	long <b>get_maxTimeOnStateB( )</b>
<b>py</b>	def <b>get_maxTimeOnStateB( )</b>
<b>cmd</b>	<b>YRelay target get_maxTimeOnStateB</b>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEB\_INVALID.

**relay->get\_module()  
relay->module()relay->get\_module()****YRelay**

Gets the `YModule` object for the device on which the function is located.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

If the function cannot be located on any module, the returned instance of `YModule` is not shown as online.

**Returns :**

an instance of `YModule`

## relay→get\_module\_async() relay→module\_async()

YRelay

Gets the YModule object for the device on which the function is located (asynchronous version).

```
js   function get_module_async( callback, context )
nodejs function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

**context** caller-specific object that is passed as-is to the callback function

### Returns :

nothing : the result is provided to the callback.

**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>
node.js	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 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>
node.js	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( ): int64</b>
vb	function <b>get_pulseTimer( ) As Long</b>
cs	long <b>get_pulseTimer( )</b>
java	long <b>get_pulseTimer( )</b>
py	def <b>get_pulseTimer( )</b>
cmd	YRelay <b>target 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\_PULSE\_TIMER\_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>
node.js	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( ): Integer</b>
vb	function <b>get_state( ) As Integer</b>
cs	int <b>get_state( )</b>
java	int <b>get_state( )</b>
py	def <b>get_state( )</b>
cmd	YRelay <b>target 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()**  
**relay→stateAtPowerOn()relay→**  
**get\_stateAtPowerOn( )**

**YRelay**

Returns the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no change).

<b>js</b>	function <b>get_stateAtPowerOn( )</b>
<b>nodejs</b>	function <b>get_stateAtPowerOn( )</b>
<b>php</b>	function <b>get_stateAtPowerOn( )</b>
<b>cpp</b>	Y_STATEATPOWERON_enum <b>get_stateAtPowerOn( )</b>
<b>m</b>	-(Y_STATEATPOWERON_enum) <b>stateAtPowerOn</b>
<b>pas</b>	function <b>get_stateAtPowerOn( )</b> : Integer
<b>vb</b>	function <b>get_stateAtPowerOn( )</b> As Integer
<b>cs</b>	int <b>get_stateAtPowerOn( )</b>
<b>java</b>	int <b>get_stateAtPowerOn( )</b>
<b>py</b>	def <b>get_stateAtPowerOn( )</b>
<b>cmd</b>	<b>YRelay target 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)**  
**relay→userData()relay→get(userData)****YRelay**

Returns the value of the userData attribute, as previously stored using method set(userData).

```
js function get(userData) 
node.js function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

**Returns :**

the object stored previously by the caller.

**relay→isOnline()****YRelay**

Checks if the relay is currently reachable, without raising any error.

js	function <b>isOnline( )</b>
node.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 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 load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network 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()**

YRelay

Sets the relay to output B (active) for a specified duration, then brings it automatically back to output A (idle state).

js	function pulse( ms_duration)
nodejs	function pulse( ms_duration)
php	function pulse( \$ms_duration)
cpp	int pulse( int ms_duration)
m	-(int) pulse : (int) ms_duration
pas	function pulse( ms_duration: LongInt): integer
vb	function pulse( ByVal ms_duration As Integer) As Integer
cs	int pulse( int ms_duration)
java	int pulse( int ms_duration)
py	def pulse( ms_duration)
cmd	YRelay target pulse ms_duration

**Parameters :**

**ms\_duration** pulse duration, in millisecondes

**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 registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php  function registerValueCallback( $callback)
cpp   int registerValueCallback( YRelayValueCallback callback)
m    -(int) registerValueCallback : (YRelayValueCallback) callback
pas   function registerValueCallback( callback: TYRelayValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs   int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
py    def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

**Parameters :**

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

**relay→set\_logicalName()**  
**relay→setLogicalName()** relay→  
**set\_logicalName( )**

YRelay

Changes the logical name of the relay.

js	function set_logicalName( newval)
nodejs	function set_logicalName( newval)
php	function set_logicalName( \$newval)
cpp	int set_logicalName( const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName( newval: string): integer
vb	function set_logicalName( ByVal newval As String) As Integer
cs	int set_logicalName( string newval)
java	int set_logicalName( String newval)
py	def set_logicalName( newval)
cmd	YRelay target set_logicalName newval

You can use `yCheckLogicalName( )` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash( )` method of the module if the modification must be kept.

**Parameters :**

**newval** a string corresponding to the logical name of the relay.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

**relay->set\_maxTimeOnStateA()**  
**relay->setMaxTimeOnStateA()relay->**  
**set\_maxTimeOnStateA( )**

**YRelay**


---

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( newval)</b>
node.js	function <b>setMaxTimeOnStateA( newval)</b>
php	function <b>set_maxTimeOnStateA( \$newval)</b>
cpp	int <b>set_maxTimeOnStateA( s64 newval)</b>
m	-(int) <b>setMaxTimeOnStateA : (s64) newval</b>
pas	function <b>set_maxTimeOnStateA( newval: int64): integer</b>
vb	function <b>set_maxTimeOnStateA( ByVal newval As Long) As Integer</b>
cs	int <b>set_maxTimeOnStateA( long newval)</b>
java	int <b>setMaxTimeOnStateA( long newval)</b>
py	def <b>set_maxTimeOnStateA( newval)</b>
cmd	YRelay <b>target set_maxTimeOnStateA 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()`  
`relay->setMaxTimeOnStateB()relay->`  
`set_maxTimeOnStateB( )`

YRelay

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

<code>js</code>	<code>function set_maxTimeOnStateB( newval)</code>
<code>nodejs</code>	<code>function set_maxTimeOnStateB( newval)</code>
<code>php</code>	<code>function set_maxTimeOnStateB( \$newval)</code>
<code>cpp</code>	<code>int set_maxTimeOnStateB( s64 newval)</code>
<code>m</code>	<code>-(int) setMaxTimeOnStateB : (s64) newval</code>
<code>pas</code>	<code>function set_maxTimeOnStateB( newval: int64): integer</code>
<code>vb</code>	<code>function set_maxTimeOnStateB( ByVal newval As Long) As Integer</code>
<code>cs</code>	<code>int set_maxTimeOnStateB( long newval)</code>
<code>java</code>	<code>int set_maxTimeOnStateB( long newval)</code>
<code>py</code>	<code>def set_maxTimeOnStateB( newval)</code>
<code>cmd</code>	<code>YRelay target set_maxTimeOnStateB newval</code>

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).

<b>js</b>	function <b>set_output( newval)</b>
<b>nodejs</b>	function <b>set_output( newval)</b>
<b>php</b>	function <b>set_output( \$newval)</b>
<b>cpp</b>	int <b>set_output( Y_OUTPUT_enum newval)</b>
<b>m</b>	-(int) setOutput : (Y_OUTPUT_enum) <b>newval</b>
<b>pas</b>	function <b>set_output( newval: Integer): integer</b>
<b>vb</b>	function <b>set_output( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_output( int newval)</b>
<b>java</b>	int <b>set_output( int newval)</b>
<b>py</b>	def <b>set_output( newval)</b>
<b>cmd</b>	<b>YRelay 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()**  
**relay->setState()|relay->set\_state( )****YRelay**

Changes the state of the relays (A for the idle position, B for the active position).

js	function <b>set_state( newval)</b>
node.js	function <b>set_state( newval)</b>
php	function <b>set_state( \$newval)</b>
cpp	int <b>set_state( Y_STATE_enum newval)</b>
m	-(int) setState : (Y_STATE_enum) <b>newval</b>
pas	function <b>set_state( newval: Integer): integer</b>
vb	function <b>set_state( ByVal newval As Integer) As Integer</b>
cs	int <b>set_state( int newval)</b>
java	int <b>set_state( int newval)</b>
py	def <b>set_state( newval)</b>
cmd	<b>YRelay 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()**  
**relay→setStateAtPowerOn()relay→**  
**set\_stateAtPowerOn( )**

**YRelay**

Preset the state of the relays at device startup (A for the idle position, B for the active position, UNCHANGED for no modification).

<b>js</b>	function <b>set_stateAtPowerOn( newval)</b>
<b>nodejs</b>	function <b>set_stateAtPowerOn( newval)</b>
<b>php</b>	function <b>set_stateAtPowerOn( \$newval)</b>
<b>cpp</b>	int <b>set_stateAtPowerOn( Y_STATEATPOWERON_enum newval)</b>
<b>m</b>	-(int) <b>setStateAtPowerOn : (Y_STATEATPOWERON_enum) newval</b>
<b>pas</b>	function <b>set_stateAtPowerOn( newval: Integer): integer</b>
<b>vb</b>	function <b>set_stateAtPowerOn( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_stateAtPowerOn( int newval)</b>
<b>java</b>	int <b>set_stateAtPowerOn( int newval)</b>
<b>py</b>	def <b>set_stateAtPowerOn( newval)</b>
<b>cmd</b>	<b>YRelay 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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData)</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()****YSensor****yFindSensor()yFindSensor( )**

Retrieves a sensor for a given identifier.

js	function <b>yFindSensor( func)</b>
node.js	function <b>FindSensor( func)</b>
php	function <b>yFindSensor( \$func)</b>
cpp	<b>YSensor*</b> <b>yFindSensor( string func)</b>
m	<b>+ (YSensor*) yFindSensor : (NSString*) func</b>
pas	function <b>yFindSensor( func: string): TYSensor</b>
vb	function <b>yFindSensor( ByVal func As String) As YSensor</b>
cs	<b>YSensor FindSensor( string func)</b>
java	<b>YSensor FindSensor( String func)</b>
py	<b>def FindSensor( 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() yFirstSensor()yFirstSensor( )

YSensor

Starts the enumeration of sensors currently accessible.

js	function <b>yFirstSensor()</b>
node.js	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)
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.

**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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**sensor→get\_advertisedValue()**  
**sensor→advertisedValue()sensor→**  
**get\_advertisedValue( )**

**YSensor**

Returns the current value of the sensor (no more than 6 characters).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YSensor target 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()**  
**sensor→currentRawValue()sensor→**  
**get\_currentRawValue( )**

**YSensor**

Returns the uncalibrated, unrounded raw value returned by the sensor.

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( ): double**  
**vb** function **get\_currentRawValue( ) As Double**  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YSensor **target get\_currentRawValue**

**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()**  
**sensor→currentValue()****sensor→get\_currentValue()**

**YSensor**

Returns the current value of the measure.

<b>js</b>	function <b>get_currentValue( )</b>
<b>nodejs</b>	function <b>get_currentValue( )</b>
<b>php</b>	function <b>get_currentValue( )</b>
<b>cpp</b>	double <b>get_currentValue( )</b>
<b>m</b>	-(double) <b>currentValue</b>
<b>pas</b>	function <b>get_currentValue( )</b> : double
<b>vb</b>	function <b>get_currentValue( )</b> As Double
<b>cs</b>	double <b>get_currentValue( )</b>
<b>java</b>	double <b>get_currentValue( )</b>
<b>py</b>	def <b>get_currentValue( )</b>
<b>cmd</b>	<b>YSensor target 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()**  
**sensor→errorMessage()sensor→**  
**get\_errorMessage( )**

**YSensor**

Returns the error message of the latest error with the sensor.

```
js function get_errorMessage( )
nodejs function get_errorMessage( )
php function get_errorMessage( )
cpp string get_errorMessage( )
m -(NSString*) errorMessage
pas function get_errorMessage( ): string
vb function get_errorMessage( ) As String
cs string get_errorMessage( )
java String get_errorMessage( )
py def get_errorMessage( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the sensor object

---

<b>sensor→get_errorType()</b>	<b>YSensor</b>
<b>sensor→errorType()sensor→get_errorType( )</b>	

---

Returns the numerical error code of the latest error with the sensor.

<b>js</b>	function <b>get_errorType( )</b>
<b>nodejs</b>	function <b>get_errorType( )</b>
<b>php</b>	function <b>get_errorType( )</b>
<b>cpp</b>	<b>YRETCODE get_errorType( )</b>
<b>pas</b>	function <b>get_errorType( ): YRETCODE</b>
<b>vb</b>	function <b>get_errorType( ) As YRETCODE</b>
<b>cs</b>	<b>YRETCODE get_errorType( )</b>
<b>java</b>	<b>int get_errorType( )</b>
<b>py</b>	<b>def 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 get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the 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()**  
**sensor→functionDescriptor()sensor→**  
**get\_functionDescriptor( )**

**YSensor**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) <b>functionDescriptor</b>
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<b>java</b>	<b>String get_functionDescriptor( )</b>
<b>py</b>	<b>def 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>
node.js	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( ) As String</b>
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 RELAY01-123456.relay1)

**Returns :**

a string that uniquely identifies the sensor (ex: RELAY01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**sensor→get\_highestValue()**  
**sensor→highestValue()** **sensor→get\_highestValue( )**

**YSensor**

Returns the maximal value observed for the measure since the device was started.

**js** function **get\_highestValue( )**  
**nodejs** function **get\_highestValue( )**  
**php** function **get\_highestValue( )**  
**cpp** double **get\_highestValue( )**  
**m** -(double) highestValue  
**pas** function **get\_highestValue( ): double**  
**vb** function **get\_highestValue( ) As Double**  
**cs** double **get\_highestValue( )**  
**java** double **get\_highestValue( )**  
**py** def **get\_highestValue( )**  
**cmd** YSensor **target get\_highestValue**

**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()**  
**sensor→logFrequency()sensor→**  
**get\_logFrequency( )**

**YSensor**

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	<b>YSensor target 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()**  
**sensor→logicalName()****sensor→get\_logicalName( )**

**YSensor**

Returns the logical name of the sensor.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	YSensor <b>target 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()**  
**sensor→lowestValue()****sensor→get\_lowestValue( )**

**YSensor**

Returns the minimal value observed for the measure since the device was started.

<b>js</b>	function <b>get_lowestValue( )</b>
<b>nodejs</b>	function <b>get_lowestValue( )</b>
<b>php</b>	function <b>get_lowestValue( )</b>
<b>cpp</b>	double <b>get_lowestValue( )</b>
<b>m</b>	-(double) lowestValue
<b>pas</b>	function <b>get_lowestValue( )</b> : double
<b>vb</b>	function <b>get_lowestValue( )</b> As Double
<b>cs</b>	double <b>get_lowestValue( )</b>
<b>java</b>	double <b>get_lowestValue( )</b>
<b>py</b>	def <b>get_lowestValue( )</b>
<b>cmd</b>	<b>YSensor target 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()**  
**sensor→module()sensor→get\_module()**

**YSensor**

Gets the `YModule` object for the device on which the function is located.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

If the function cannot be located on any module, the returned instance of `YModule` is not shown as online.

**Returns :**

an instance of `YModule`

**sensor→get\_module\_async()**  
**sensor→module\_async()****YSensor**

Gets the YModule object for the device on which the function is located (asynchronous version).

```
js   function get_module_async( callback, context )
nodejs function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**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( startTime, endTime)</b>
nodejs	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	YDataSet <b>get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
java	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
py	def <b>get_recordedData( startTime, endTime)</b>
cmd	YSensor <b>target get_recordedData startTime 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 measure, 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 measure, 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 get_reportFrequency( )
nodejs function get_reportFrequency( )
php  function get_reportFrequency( )
cpp   string get_reportFrequency( )
m    -(NSString*) reportFrequency
pas   function get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs    string get_reportFrequency( )
java  String get_reportFrequency( )
py    def get_reportFrequency( )
cmd   YSensor target get_reportFrequency
```

**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 get_resolution( )
node.js function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YSensor target get_resolution
```

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*) <b>unit</b>
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 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 get(userData) 
node.js function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

**Returns :**

the object stored previously by the caller.

**sensor→isOnline()sensor→isOnline( )****YSensor**

Checks if the sensor is currently reachable, without raising any error.

js	function <b>isOnline( )</b>
node.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 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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-(YRETCODE) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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 <b>loadCalibrationPoints( rawValues, refValues )</b>
nodejs	function <b>loadCalibrationPoints( rawValues, refValues )</b>
php	function <b>loadCalibrationPoints( &amp;\$rawValues, &amp;\$refValues )</b>
cpp	int <b>loadCalibrationPoints( vector&lt;double&gt;&amp; rawValues, vector&lt;double&gt;&amp; refValues )</b>
m	- <b>(int) loadCalibrationPoints : (NSMutableArray*) rawValues : (NSMutableArray*) refValues</b>
pas	function <b>loadCalibrationPoints( var rawValues: TDoubleArray, var refValues: TDoubleArray): LongInt</b>
vb	procedure <b>loadCalibrationPoints( )</b>
cs	int <b>loadCalibrationPoints( List&lt;double&gt; rawValues, List&lt;double&gt; refValues )</b>
java	int <b>loadCalibrationPoints( ArrayList&lt;Double&gt; rawValues, ArrayList&lt;Double&gt; refValues )</b>
py	def <b>loadCalibrationPoints( rawValues, refValues )</b>
cmd	YSensor target <b>loadCalibrationPoints rawValues refValues</b>

**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.

---

<b>sensor→registerTimedReportCallback()</b>	<b>sensor→registerTimedReportCallback( )</b>	<b>YSensor</b>
---	--	----------------

---

Registers the callback function that is invoked on every periodic timed notification.

<b>js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>node.js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>php</b>	function registerTimedReportCallback( <b>\$callback</b> )
<b>cpp</b>	int registerTimedReportCallback( YSensorTimedReportCallback <b>callback</b> )
<b>m</b>	-(int) registerTimedReportCallback : (YSensorTimedReportCallback) <b>callback</b>
<b>pas</b>	function registerTimedReportCallback( <b>callback</b> : TYSensorTimedReportCallback): LongInt
<b>vb</b>	function registerTimedReportCallback( ) As Integer
<b>cs</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>java</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>py</b>	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YSensorValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YSensorValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYSensorValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**sensor→setHighestValue()** sensor→  
**set\_highestValue()**

**YSensor**

Changes the recorded maximal value observed.

<b>js</b>	function <b>set_highestValue( newval)</b>
<b>nodejs</b>	function <b>set_highestValue( newval)</b>
<b>php</b>	function <b>set_highestValue( \$newval)</b>
<b>cpp</b>	int <b>set_highestValue( double newval)</b>
<b>m</b>	-(int) setHighestValue : (double) <b>newval</b>
<b>pas</b>	function <b>set_highestValue( newval: double): integer</b>
<b>vb</b>	function <b>set_highestValue( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_highestValue( double newval)</b>
<b>java</b>	int <b>set_highestValue( double newval)</b>
<b>py</b>	def <b>set_highestValue( newval)</b>
<b>cmd</b>	<b>YSensor 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.

**sensor→set\_logFrequency()**  
**sensor→setLogFrequency()** sensor→  
**set\_logFrequency( )**

**YSensor**

Changes the datalogger recording frequency for this function.

```
js   function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php  function set_logFrequency( $newval)
cpp   int set_logFrequency( const string& newval)
m    -(int) setLogFrequency : (NSString*) newval
pas   function set_logFrequency( newval: string): integer
vb    function set_logFrequency( ByVal newval As String) As Integer
cs    int set_logFrequency( string newval)
java  int set_logFrequency( String newval)
py    def set_logFrequency( newval)
cmd   YSensor target set_logFrequency 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 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()**  
**sensor→setLogicalName()****sensor→**  
**set\_logicalName( )**

**YSensor**

Changes the logical name of the sensor.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YSensor 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 sensor.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

**sensor→set\_lowestValue()**  
**sensor→setLowestValue()** sensor→  
**set\_lowestValue()**

YSensor

Changes the recorded minimal value observed.

```
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 YSensor target set_lowestValue newval
```

**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.

<b>js</b>	function <b>set_reportFrequency( newval)</b>
<b>nodejs</b>	function <b>set_reportFrequency( newval)</b>
<b>php</b>	function <b>set_reportFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_reportFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setReportFrequency : (NSString*) newval</b>
<b>pas</b>	function <b>set_reportFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_reportFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_reportFrequency( string newval)</b>
<b>java</b>	int <b>set_reportFrequency( String newval)</b>
<b>py</b>	def <b>set_reportFrequency( newval)</b>
<b>cmd</b>	<b>YSensor 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.

<b>sensor→set_resolution()</b>	<b>YSensor</b>
<b>sensor→setResolution()</b>	<b>sensor→</b>
<b>set_resolution()</b>	

Changes the resolution of the measured physical values.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YSensor target set_resolution newval
```

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.

---

<b>sensor→set(userData)</b>	<b>YSensor</b>
<b>sensor→setUserData()sensor→set(userData()</b>	

---

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData( data)</b>
nodejs	function <b>set(userData( data)</b>
php	function <b>set(userData( \$data)</b>
cpp	void <b>set(userData( void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData( data: Tobject)</b>
vb	procedure <b>set(userData( ByVal data As Object)</b>
cs	void <b>set(userData( object data)</b>
java	void <b>set(userData( Object data)</b>
py	def <b>set(userData( 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>
node.js	var yoctolib = require('yoctolib');
	var YServo = yoctolib.YServo;
php	require_once('yocto_servo.php');
cpp	#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.

### 3. Reference

#### **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()yFindServo( )

YServo

Retrieves a servo for a given identifier.

```
js function yFindServo( func)
node.js function FindServo( func)
php function yFindServo( $func)
cpp YServo* yFindServo( const string& func)
m YServo* yFindServo( NSString* func)
pas function yFindServo( func: string): TYServo
vb function yFindServo( ByVal func As String) As YServo
cs YServo FindServo( string func)
java YServo FindServo( String func)
def FindServo( 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 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()****YServo****yFirstServo()yFirstServo( )**

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()****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	- <b>(NSString*) describe</b>
pas	function <b>describe( ): string</b>
vb	function <b>describe( ) As String</b>
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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)=RELAYL01-123456.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 servo (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**servo→get\_advertisedValue()**  
**servo→advertisedValue()servo→**  
**get\_advertisedValue( )**

YServo

Returns the current value of the servo (no more than 6 characters).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
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 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()****YServo**

Returns the state of the servos.

```
js function get_enabled( )
node.js function get_enabled( )
php function get_enabled( )
cpp Y_ENABLED_enum get_enabled( )
m -(Y_ENABLED_enum) enabled
pas function get_enabled( ): Integer
vb function get_enabled( ) As Integer
cs int get_enabled( )
java int get_enabled( )
py def get_enabled( )
cmd YServo target get_enabled
```

**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()**  
**servo→enabledAtPowerOn()servo→**  
**get\_enabledAtPowerOn( )**

**YServo**

Returns the servo signal generator state at power up.

<b>js</b>	function <b>get_enabledAtPowerOn( )</b>
<b>nodejs</b>	function <b>get_enabledAtPowerOn( )</b>
<b>php</b>	function <b>get_enabledAtPowerOn( )</b>
<b>cpp</b>	Y_ENABLEDATPOWERON_enum <b>get_enabledAtPowerOn( )</b>
<b>m</b>	-(Y_ENABLEDATPOWERON_enum) enabledAtPowerOn
<b>pas</b>	function <b>get_enabledAtPowerOn( ): Integer</b>
<b>vb</b>	function <b>get_enabledAtPowerOn( ) As Integer</b>
<b>cs</b>	int <b>get_enabledAtPowerOn( )</b>
<b>java</b>	int <b>get_enabledAtPowerOn( )</b>
<b>py</b>	def <b>get_enabledAtPowerOn( )</b>
<b>cmd</b>	<b>YServo target 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→getErrorMessage()**  
**servo→errorMessage()**servo→  
**getErrorMessage( )**

**YServo**

Returns the error message of the latest error with the servo.

```
js function getErrorMessage( )
nodejs function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ): string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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()**  
**servo→friendlyName()servo→**  
**get\_friendlyName( )**

**YServo**

Returns a global identifier of the servo in the format MODULE\_NAME . FUNCTION\_NAME.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the 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.

<b>js</b>	function <b>get_functionDescriptor( )</b>
<b>nodejs</b>	function <b>get_functionDescriptor( )</b>
<b>php</b>	function <b>get_functionDescriptor( )</b>
<b>cpp</b>	<b>YFUN_DESCR get_functionDescriptor( )</b>
<b>m</b>	-(YFUN_DESCR) functionDescriptor
<b>pas</b>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<b>vb</b>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<b>cs</b>	<b>YFUN_DESCR get_functionDescriptor( )</b>
<b>java</b>	<b>String get_functionDescriptor( )</b>
<b>py</b>	<b>def 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>
node.js	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( ) As String</b>
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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the servo (ex: RELAYL01-123456.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 get_logicalName( )
node.js function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YServo target get_logicalName
```

**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 online.

**Returns :**

an instance of YModule

**servo→get\_module\_async()**  
**servo→module\_async()****YServo**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

```
js   function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**servo→get\_neutral()**  
**servo→neutral()servo→get\_neutral( )****YServo**

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) <b>neutral</b>
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	<b>YServo target 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()  
servo→position()servo→get\_position()****YServo**

Returns the current servo position.

```
js function get_position( )
node.js function get_position( )
php function get_position( )
cpp int get_position( )
m -(int) position
pas function get_position( ): LongInt
vb function get_position( ) As Integer
cs int get_position( )
java int get_position( )
py def get_position( )
cmd YServo target get_position
```

**Returns :**

an integer corresponding to the current servo position

On failure, throws an exception or returns Y\_POSITION\_INVALID.

**servo→get\_positionAtPowerOn()**  
**servo→positionAtPowerOn()servo→**  
**get\_positionAtPowerOn( )**

YServo

Returns the servo position at device power up.

<b>js</b>	function <b>get_positionAtPowerOn( )</b>
<b>node.js</b>	function <b>get_positionAtPowerOn( )</b>
<b>php</b>	function <b>get_positionAtPowerOn( )</b>
<b>cpp</b>	int <b>get_positionAtPowerOn( )</b>
<b>m</b>	-(int) <b>positionAtPowerOn</b>
<b>pas</b>	function <b>get_positionAtPowerOn( )</b> : LongInt
<b>vb</b>	function <b>get_positionAtPowerOn( )</b> As Integer
<b>cs</b>	int <b>get_positionAtPowerOn( )</b>
<b>java</b>	int <b>get_positionAtPowerOn( )</b>
<b>py</b>	def <b>get_positionAtPowerOn( )</b>
<b>cmd</b>	YServo <b>target 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( )**

YServo

Returns the current range of use of the servo.

```
js function get_range( )
node.js function get_range( )
php function get_range( )
cpp int get_range( )
m -(int) range
pas function get_range( ): LongInt
vb function get_range( ) As Integer
cs int get_range( )
java int get_range( )
py def get_range( )
cmd YServo target get_range
```

**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→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.

**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	- <b>(BOOL) 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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-( <b>YRETCODE</b> ) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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 move( target, ms_duration)
nodejs function move( target, ms_duration)
php function move( $target, $ms_duration)
cpp int move( int target, int ms_duration)
m -(int) move : (int) target : (int) ms_duration
pas function move( target: LongInt, ms_duration: LongInt): integer
vb function move( ByVal target As Integer,
                  ByVal ms_duration As Integer) As Integer
cs int move( int target, int ms_duration)
java int move( int target, int ms_duration)
py def move( target, ms_duration)
cmd YServo target move target ms_duration
```

**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()****YServo**

Continues the enumeration of servos started using `yFirstServo()`.

js	function <b>nextServo()</b>
node.js	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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YServoValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YServoValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYServoValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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( newval)</b>
nodejs	function <b>set_enabled( newval)</b>
php	function <b>set_enabled( \$newval)</b>
cpp	int <b>set_enabled( Y_ENABLED_enum newval)</b>
m	-(int) setEnabled : (Y_ENABLED_enum) <b>newval</b>
pas	function <b>set_enabled( newval: Integer): integer</b>
vb	function <b>set_enabled( ByVal newval As Integer) As Integer</b>
cs	int <b>set_enabled( int newval)</b>
java	int <b>set_enabled( int newval)</b>
py	def <b>set_enabled( newval)</b>
cmd	YServo target <b>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()**  
**servo→setEnabledAtPowerOn()servo→**  
**set\_enabledAtPowerOn( )**

**YServo**

Configure the servo signal generator state at power up.

<b>js</b>	function <b>set_enabledAtPowerOn( newval)</b>
<b>nodejs</b>	function <b>set_enabledAtPowerOn( newval)</b>
<b>php</b>	function <b>set_enabledAtPowerOn( \$newval)</b>
<b>cpp</b>	int <b>set_enabledAtPowerOn( Y_ENABLEDATPOWERON_enum newval)</b>
<b>m</b>	-(int) <b>setEnabledAtPowerOn : (Y_ENABLEDATPOWERON_enum) newval</b>
<b>pas</b>	function <b>set_enabledAtPowerOn( newval: Integer): integer</b>
<b>vb</b>	function <b>set_enabledAtPowerOn( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_enabledAtPowerOn( int newval)</b>
<b>java</b>	int <b>set_enabledAtPowerOn( int newval)</b>
<b>py</b>	def <b>set_enabledAtPowerOn( newval)</b>
<b>cmd</b>	<b>YServo target set_enabledAtPowerOn 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()**  
**servo→setLogicalName()** servo→  
**set\_logicalName( )**

YServo

Changes the logical name of the servo.

js	function <b>set_logicalName( newval)</b>
node.js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	YServo <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 servo.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

**servo→set\_neutral()** YServo  
**servo→setNeutral()** **servo→set\_neutral( )**

Changes the duration of the pulse corresponding to the neutral position of the servo.

js	function <b>set_neutral( newval)</b>
node.js	function <b>set_neutral( newval)</b>
php	function <b>set_neutral( \$newval)</b>
cpp	int <b>set_neutral( int newval)</b>
m	-(int) setNeutral : (int) newval
pas	function <b>set_neutral( newval: LongInt): integer</b>
vb	function <b>set_neutral( ByVal newval As Integer) As Integer</b>
cs	int <b>set_neutral( int newval)</b>
java	int <b>set_neutral( int newval)</b>
py	def <b>set_neutral( newval)</b>
cmd	<b>YServo 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 set_position( newval)`

`nodejs function set_position( newval)`

`php function set_position( $newval)`

`cpp int set_position( int newval)`

`m -(int) setPosition : (int) newval`

`pas function set_position( newval: LongInt): integer`

`vb function set_position( ByVal newval As Integer) As Integer`

`cs int set_position( int newval)`

`java int set_position( int newval)`

`py def set_position( newval)`

`cmd YServo target set_position newval`

**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()**  
**servo→setPositionAtPowerOn()servo→**  
**set\_positionAtPowerOn( )**

YServo

Configure the servo position at device power up.

```
js function set_positionAtPowerOn( newval)
nodejs function set_positionAtPowerOn( newval)
php function set_positionAtPowerOn( $newval)
cpp int set_positionAtPowerOn( int newval)
m -(int) setPositionAtPowerOn : (int) newval
pas function set_positionAtPowerOn( newval: LongInt): integer
vb function set_positionAtPowerOn( ByVal newval As Integer) As Integer
cs int set_positionAtPowerOn( int newval)
java int set_positionAtPowerOn( int newval)
py def set_positionAtPowerOn( newval)
cmd YServo target set_positionAtPowerOn newval
```

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.

<b>js</b>	function <b>set_range( newval)</b>
<b>nodejs</b>	function <b>set_range( newval)</b>
<b>php</b>	function <b>set_range( \$newval)</b>
<b>cpp</b>	int <b>set_range( int newval)</b>
<b>m</b>	-(int) setRange : (int) <b>newval</b>
<b>pas</b>	function <b>set_range( newval: LongInt): integer</b>
<b>vb</b>	function <b>set_range( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_range( int newval)</b>
<b>java</b>	int <b>set_range( int newval)</b>
<b>py</b>	def <b>set_range( newval)</b>
<b>cmd</b>	<b>YServo 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()****YServo**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b> void* <b>data</b>
m	-(void) <b>setUserData</b> : (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');
cpp #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()****YTemperature****yFindTemperature()yFindTemperature( )**

Retrieves a temperature sensor for a given identifier.

<code>js</code>	<code>function yFindTemperature( func)</code>
<code>node.js</code>	<code>function FindTemperature( func)</code>
<code>php</code>	<code>function yFindTemperature( \$func)</code>
<code>cpp</code>	<code>YTemperature* yFindTemperature( const string&amp; func)</code>
<code>m</code>	<code>YTemperature* yFindTemperature( NSString* func)</code>
<code>pas</code>	<code>function yFindTemperature( func: string): TYTemperature</code>
<code>vb</code>	<code>function yFindTemperature( ByVal func As String) As YTemperature</code>
<code>cs</code>	<code>YTemperature FindTemperature( string func)</code>
<code>java</code>	<code>YTemperature FindTemperature( String func)</code>
<code>py</code>	<code>def FindTemperature( 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 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() yFirstTemperature()yFirstTemperature( )

YTemperature

Starts the enumeration of temperature sensors currently accessible.

```
js function yFirstTemperature( )
node.js function FirstTemperature( )
php function yFirstTemperature( )
cpp YTemperature* yFirstTemperature( )
m YTemperature* yFirstTemperature( )
pas function yFirstTemperature( ): TYTemperature
vb function yFirstTemperature( ) As YTemperature
cs YTemperature FirstTemperature( )
java YTemperature FirstTemperature( )
py def FirstTemperature( )
```

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)
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  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()****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	- <b>(NSString*) describe</b>
pas	function <b>describe( ): string</b>
vb	function <b>describe( ) As String</b>
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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)=RELAYL01-123456.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 temperature sensor (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**temperature→get\_advertisedValue()****YTemperature****temperature→advertisedValue()temperature→  
get\_advertisedValue( )**

Returns the current value of the temperature sensor (no more than 6 characters).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>node.js</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YTemperature target 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()**  
**temperature→currentRawValue()** **temperature→get\_currentRawValue( )**

**YTemperature**

Returns the uncalibrated, unrounded raw value returned by the sensor.

```
js function get_currentRawValue( )
nodejs function get_currentRawValue( )
php function get_currentRawValue( )
cpp double get_currentRawValue( )
m -(double) currentRawValue
pas function get_currentRawValue( ): double
vb function get_currentRawValue( ) As Double
cs double get_currentRawValue( )
java double get_currentRawValue( )
py def get_currentRawValue( )
cmd YTemperature target get_currentRawValue
```

**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()**  
**temperature→currentValue()temperature→**  
**get\_currentValue()**

**YTemperature**

Returns the current value of the temperature.

<b>js</b>	function <b>get_currentValue( )</b>
<b>nodejs</b>	function <b>get_currentValue( )</b>
<b>php</b>	function <b>get_currentValue( )</b>
<b>cpp</b>	double <b>get_currentValue( )</b>
<b>m</b>	-(double) <b>currentValue</b>
<b>pas</b>	function <b>get_currentValue( )</b> : double
<b>vb</b>	function <b>get_currentValue( )</b> As Double
<b>cs</b>	double <b>get_currentValue( )</b>
<b>java</b>	double <b>get_currentValue( )</b>
<b>py</b>	def <b>get_currentValue( )</b>
<b>cmd</b>	<b>YTemperature target 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→getErrorMessage()**  
**temperature→errorMessage()** **temperature→getErrorMessage( )**

**YTemperature**

Returns the error message of the latest error with the temperature sensor.

```
js function getErrorMessage( )
nodejs function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ): string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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

---

<b>temperature→get_errorType()</b>	<b>YTemperature</b>
<b>temperature→errorType()</b> <b>temperature→get_errorType( )</b>	

Returns the numerical error code of the latest error with the temperature sensor.

```
js    function get_errorType( )
nodejs function get_errorType( )
php   function get_errorType( )
cpp   YRETCODE get_errorType( )
pas   function get_errorType( ): YRETCODE
vb    function get_errorType( ) As YRETCODE
cs    YRETCODE get_errorType( )
java  int get_errorType( )
py    def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a number corresponding to the code of the latest error that 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 get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

The returned string uses the logical names of the module and of the temperature sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the temperature sensor (for exemple: 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.

<code>js</code>	<code>function get_functionDescriptor( )</code>
<code>node.js</code>	<code>function get_functionDescriptor( )</code>
<code>php</code>	<code>function get_functionDescriptor( )</code>
<code>cpp</code>	<code>YFUN_DESCR get_functionDescriptor( )</code>
<code>m</code>	<code>-(YFUN_DESCR) functionDescriptor</code>
<code>pas</code>	<code>function get_functionDescriptor( ): YFUN_DESCR</code>
<code>vb</code>	<code>function get_functionDescriptor( ) As YFUN_DESCR</code>
<code>cs</code>	<code>YFUN_DESCR get_functionDescriptor( )</code>
<code>java</code>	<code>String get_functionDescriptor( )</code>
<code>py</code>	<code>def get_functionDescriptor( )</code>

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()**  
**temperature→functionId()** **temperature→get\_functionId( )**

**YTemperature**

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 get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the temperature sensor. (for example RELAY01-123456.relay1)

**Returns :**

a string that uniquely identifies the temperature sensor (ex: RELAY01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**temperature→get\_highestValue()**  
**temperature→highestValue()** **temperature→get\_highestValue( )**

**YTemperature**

Returns the maximal value observed for the temperature since the device was started.

**js** function **get\_highestValue( )**  
**nodejs** function **get\_highestValue( )**  
**php** function **get\_highestValue( )**  
**cpp** double **get\_highestValue( )**  
**m** -(double) highestValue  
**pas** function **get\_highestValue( )**: double  
**vb** function **get\_highestValue( )** As Double  
**cs** double **get\_highestValue( )**  
**java** double **get\_highestValue( )**  
**py** def **get\_highestValue( )**  
**cmd** YTemperature **target get\_highestValue**

**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>
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	<b>YTemperature target 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()**  
**temperature→logicalName()temperature→**  
**get\_logicalName( )**

**YTemperature**

Returns the logical name of the temperature sensor.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	<b>YTemperature target 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()**  
**temperature→lowestValue()** **temperature→get\_lowestValue( )**

**YTemperature**

Returns the minimal value observed for the temperature since the device was started.

<b>js</b>	function <b>get_lowestValue( )</b>
<b>nodejs</b>	function <b>get_lowestValue( )</b>
<b>php</b>	function <b>get_lowestValue( )</b>
<b>cpp</b>	double <b>get_lowestValue( )</b>
<b>m</b>	-(double) lowestValue
<b>pas</b>	function <b>get_lowestValue( ): double</b>
<b>vb</b>	function <b>get_lowestValue( ) As Double</b>
<b>cs</b>	double <b>get_lowestValue( )</b>
<b>java</b>	double <b>get_lowestValue( )</b>
<b>py</b>	def <b>get_lowestValue( )</b>
<b>cmd</b>	<b>YTemperature target 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()**  
**temperature→module()temperature→**  
**get\_module( )**

**YTemperature**

Gets the **YModule** object for the device on which the function is located.

<b>js</b>	function <b>get_module( )</b>
<b>nodejs</b>	function <b>get_module( )</b>
<b>php</b>	function <b>get_module( )</b>
<b>cpp</b>	<b>YModule * get_module( )</b>
<b>m</b>	-(YModule*) module
<b>pas</b>	function <b>get_module( )</b> : TYModule
<b>vb</b>	function <b>get_module( )</b> As YModule
<b>cs</b>	<b>YModule get_module( )</b>
<b>java</b>	<b>YModule get_module( )</b>
<b>py</b>	<b>def get_module( )</b>

If the function cannot be located on any module, the returned instance of **YModule** is not shown as online.

**Returns :**

an instance of **YModule**

**temperature→get\_module\_async()**  
**temperature→module\_async()****YTemperature**

Gets the YModule object for the device on which the function is located (asynchronous version).

```
js   function get_module_async( callback, context )
nodejs function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**temperature→get\_recordedData()** YTemperature

**temperature→recordedData()** **temperature→get\_recordedData()**

---

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

<code>js</code>	<code>function get_recordedData( startTime, endTime)</code>
<code>nodejs</code>	<code>function get_recordedData( startTime, endTime)</code>
<code>php</code>	<code>function get_recordedData( \$startTime, \$endTime)</code>
<code>cpp</code>	<code>YDataSet get_recordedData( s64 startTime, s64 endTime)</code>
<code>m</code>	<code>-(YDataSet*) recordedData : (s64) startTime : (s64) endTime</code>
<code>pas</code>	<code>function get_recordedData( startTime: int64, endTime: int64): TYDataSet</code>
<code>vb</code>	<code>function get_recordedData( ) As YDataSet</code>
<code>cs</code>	<code>YDataSet get_recordedData( long startTime, long endTime)</code>
<code>java</code>	<code>YDataSet get_recordedData( long startTime, long endTime)</code>
<code>py</code>	<code>def get_recordedData( startTime, endTime)</code>
<code>cmd</code>	<code>YTemperature target get_recordedData startTime endTime</code>

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 measure, 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 measure, 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()**  
**temperature→reportFrequency()temperature→**  
**get\_reportFrequency( )**

**YTemperature**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

```
js   function get_reportFrequency( )
nodejs function get_reportFrequency( )
php  function get_reportFrequency( )
cpp   string get_reportFrequency( )
m    -(NSString*) reportFrequency
pas   function get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs   string get_reportFrequency( )
java  String get_reportFrequency( )
py    def get_reportFrequency( )
cmd   YTemperature target get_reportFrequency
```

**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()**  
**temperature→resolution()** **temperature→get\_resolution()**

**YTemperature**

Returns the resolution of the measured values.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YTemperature target get_resolution
```

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()**  
**temperature→sensorType()** **temperature→get\_sensorType( )**

**YTemperature**

Returns the temperature sensor type.

<b>js</b>	function <b>get_sensorType( )</b>
<b>nodejs</b>	function <b>get_sensorType( )</b>
<b>php</b>	function <b>get_sensorType( )</b>
<b>cpp</b>	<b>Y_SENSORTYPE_enum get_sensorType( )</b>
<b>m</b>	-( <b>Y_SENSORTYPE_enum</b> ) <b>sensorType</b>
<b>pas</b>	function <b>get_sensorType( )</b> : Integer
<b>vb</b>	function <b>get_sensorType( )</b> As Integer
<b>cs</b>	<b>int get_sensorType( )</b>
<b>java</b>	<b>int get_sensorType( )</b>
<b>py</b>	<b>def get_sensorType( )</b>
<b>cmd</b>	<b>YTemperature target 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>
node.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( ): string</b>
vb	function <b>get_unit( ) As String</b>
cs	string <b>get_unit( )</b>
java	<b>String get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YTemperature target 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>
node.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.

**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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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	<code>function nextTemperature( )</code>
nodejs	<code>function nextTemperature( )</code>
php	<code>function nextTemperature( )</code>
cpp	<code>YTemperature * nextTemperature( )</code>
m	<code>-(YTemperature*) nextTemperature</code>
pas	<code>function nextTemperature( ): TYTemperature</code>
vb	<code>function nextTemperature( ) As YTemperature</code>
cs	<code>YTemperature nextTemperature( )</code>
java	<code>YTemperature nextTemperature( )</code>
py	<code>def nextTemperature( )</code>

**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 registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php  function registerTimedReportCallback( $callback)
cpp   int registerTimedReportCallback( YTemperatureTimedReportCallback callback)
m     -(int) registerTimedReportCallback : (YTemperatureTimedReportCallback) callback
pas   function registerTimedReportCallback( callback: TYTemperatureTimedReportCallback): LongInt
vb    function registerTimedReportCallback( ) As Integer
cs    int registerTimedReportCallback( TimedReportCallback callback)
java  int registerTimedReportCallback( TimedReportCallback callback)
py    def registerTimedReportCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

**Parameters :**

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and 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 registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php  function registerValueCallback( $callback)
cpp   int registerValueCallback( YTemperatureValueCallback callback)
m    -(int) registerValueCallback : (YTemperatureValueCallback) callback
pas   function registerValueCallback( callback: TYTemperatureValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs   int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
py    def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

#### Parameters :

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

**temperature→set\_highestValue()**  
**temperature→setHighestValue()** **temperature→set\_highestValue( )**

**YTemperature**

Changes the recorded maximal value observed.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YTemperature target set_highestValue newval
```

**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.

<b>js</b>	function <b>set_logFrequency( newval)</b>
<b>node.js</b>	function <b>set_logFrequency( newval)</b>
<b>php</b>	function <b>set_logFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_logFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogFrequency : (NSString*) newval</b>
<b>pas</b>	function <b>set_logFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logFrequency( string newval)</b>
<b>java</b>	int <b>set_logFrequency( String newval)</b>
<b>py</b>	<b>def set_logFrequency( newval)</b>
<b>cmd</b>	<b>YTemperature 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.

**temperature→set\_logicalName()**  
**temperature→setLogicalName()** **temperature→**  
**set\_logicalName( )**

**YTemperature**

Changes the logical name of the temperature sensor.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YTemperature target set_logicalName newval
```

You can use `yCheckLogicalName( )` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash( )` method of the module if the modification must be kept.

**Parameters :**

**newval** a string corresponding to the logical name of the temperature sensor.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

**temperature→set\_lowestValue()**  
**temperature→setLowestValue()** **temperature→**  
**set\_lowestValue( )**

**YTemperature**

Changes the recorded minimal value observed.

<b>js</b>	function <b>set_lowestValue( newval)</b>
<b>nodejs</b>	function <b>set_lowestValue( newval)</b>
<b>php</b>	function <b>set_lowestValue( \$newval)</b>
<b>cpp</b>	int <b>set_lowestValue( double newval)</b>
<b>m</b>	-(int) <b>setLowestValue : (double) newval</b>
<b>pas</b>	function <b>set_lowestValue( newval: double): integer</b>
<b>vb</b>	function <b>set_lowestValue( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_lowestValue( double newval)</b>
<b>java</b>	int <b>set_lowestValue( double newval)</b>
<b>py</b>	def <b>set_lowestValue( newval)</b>
<b>cmd</b>	<b>YTemperature 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.

**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 set_resolution( newval)
node.js function set_resolution( newval)
php  function set_resolution( $newval)
cpp   int set_resolution( double newval)
m    -(int) setResolution : (double) newval
pas   function set_resolution( newval: double): integer
vb    function set_resolution( ByVal newval As Double) As Integer
cs    int set_resolution( double newval)
java  int set_resolution( double newval)
py    def set_resolution( newval)
cmd   YTemperature target set_resolution newval
```

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()**  
**temperature→setSensorType()** **temperature→**  
**set\_sensorType( )**

**YTemperature**

Modify the temperature sensor type.

<b>js</b>	function <b>set_sensorType( newval)</b>
<b>nodejs</b>	function <b>set_sensorType( newval)</b>
<b>php</b>	function <b>set_sensorType( \$newval)</b>
<b>cpp</b>	int <b>set_sensorType( Y_SENSORTYPE_enum newval)</b>
<b>m</b>	- (int) <b>setSensorType : (Y_SENSORTYPE_enum) newval</b>
<b>pas</b>	function <b>set_sensorType( newval: Integer): integer</b>
<b>vb</b>	function <b>set_sensorType( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_sensorType( int newval)</b>
<b>java</b>	int <b>set_sensorType( int newval)</b>
<b>py</b>	def <b>set_sensorType( newval)</b>
<b>cmd</b>	<b>YTemperature target set_sensorType newval</b>

This function is used 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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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	<script type='text/javascript' src='yocto_tilt.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YTilt = yoctolib.YTilt;
php	require_once('yocto_tilt.php');
cpp	#include "yocto_tilt.h"
m	#import "yocto_tilt.h"
pas	uses yocto_tilt;
vb	yocto_tilt.vb
cs	yocto_tilt.cs
java	import com.yoctopuce.YoctoAPI.YTilt;
py	from yocto_tilt import *

### 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.

<b>tilt→get_highestValue()</b>	Returns the maximal value observed for the inclination since the device was started.
<b>tilt→get_logFrequency()</b>	Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>tilt→get_logicalName()</b>	Returns the logical name of the tilt sensor.
<b>tilt→get_lowestValue()</b>	Returns the minimal value observed for the inclination since the device was started.
<b>tilt→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>tilt→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>tilt→get_recordedData(startTime, endTime)</b>	Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>tilt→get_reportFrequency()</b>	Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>tilt→get_resolution()</b>	Returns the resolution of the measured values.
<b>tilt→get_unit()</b>	Returns the measuring unit for the inclination.
<b>tilt→get(userData)</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>tilt→isOnline()</b>	Checks if the tilt sensor is currently reachable, without raising any error.
<b>tilt→isOnline_async(callback, context)</b>	Checks if the tilt sensor is currently reachable, without raising any error (asynchronous version).
<b>tilt→load(msValidity)</b>	Preloads the tilt sensor cache with a specified validity duration.
<b>tilt→loadCalibrationPoints(rawValues, refValues)</b>	Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>tilt→load_async(msValidity, callback, context)</b>	Preloads the tilt sensor cache with a specified validity duration (asynchronous version).
<b>tilt→nextTilt()</b>	Continues the enumeration of tilt sensors started using yFirstTilt( ).
<b>tilt→registerTimedReportCallback(callback)</b>	Registers the callback function that is invoked on every periodic timed notification.
<b>tilt→registerValueCallback(callback)</b>	Registers the callback function that is invoked on every change of advertised value.
<b>tilt→set_highestValue(newval)</b>	Changes the recorded maximal value observed.
<b>tilt→set_logFrequency(newval)</b>	Changes the datalogger recording frequency for this function.
<b>tilt→set_logicalName(newval)</b>	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() yFindTilt()yFindTilt( )

YTilt

Retrieves a tilt sensor for a given identifier.

```
js function yFindTilt( func)
node.js function FindTilt( func)
php function yFindTilt( $func)
cpp YTilt* yFindTilt( const string& func)
m YTilt* yFindTilt( NSString* func)
pas function yFindTilt( func: string): TYTilt
vb function yFindTilt( ByVal func As String) As YTilt
cs YTilt FindTilt( string func)
java YTilt FindTilt( String func)
def FindTilt( func)
py
```

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()yFirstTilt( )**

Starts the enumeration of tilt sensors currently accessible.

js	function <b>yFirstTilt( )</b>
nodejs	function <b>FirstTilt( )</b>
php	function <b>yFirstTilt( )</b>
cpp	YTilt* <b>yFirstTilt( )</b>
m	YTilt* <b>yFirstTilt( )</b>
pas	function <b>yFirstTilt( )</b> : TYTilt
vb	function <b>yFirstTilt( )</b> As YTilt
cs	YTilt <b>FirstTilt( )</b>
java	YTilt <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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

`tilt→get_advertisedValue()`  
`tilt→advertisedValue()`  
`tilt→get_advertisedValue( )`

YTilt

Returns the current value of the tilt sensor (no more than 6 characters).

`js` `function get_advertisedValue( )`  
`nodejs` `function get_advertisedValue( )`  
`php` `function get_advertisedValue( )`  
`cpp` `string get_advertisedValue( )`  
`m` `-(NSString*) advertisedValue`  
`pas` `function get_advertisedValue( ): string`  
`vb` `function get_advertisedValue( ) As String`  
`cs` `string get_advertisedValue( )`  
`java` `String get_advertisedValue( )`  
`py` `def get_advertisedValue( )`  
`cmd` `YTilt target get_advertisedValue`

**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()  
tilt→currentRawValue()tilt→  
get\_currentRawValue( )**

YTilt

---

Returns the uncalibrated, unrounded raw value returned by the sensor.

```
js    function get_currentRawValue( )
nodejs function get_currentRawValue( )
php   function get_currentRawValue( )
cpp   double get_currentRawValue( )
m     -(double) currentRawValue
pas   function get_currentRawValue( ): double
vb    function get_currentRawValue( ) As Double
cs    double get_currentRawValue( )
java  double get_currentRawValue( )
py    def get_currentRawValue( )
cmd   YTilt target get_currentRawValue
```

**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()**

YTilt

Returns the current value of the inclination.

```
js function get_currentValue( )
node.js function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YTilt target get_currentValue
```

**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 get_errorMessage( )
nodejs function get_errorMessage( )
php  function get_errorMessage( )
cpp   string get_errorMessage( )
m    -(NSString*) errorMessage
pas   function get_errorMessage( ): string
vb    function get_errorMessage( ) As String
cs   string get_errorMessage( )
java  String get_errorMessage( )
py    def get_errorMessage( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the tilt sensor object

**tilt→get\_errorType()  
tilt→errorType()tilt→get\_errorType()****YTilt**

Returns the numerical error code of the latest error with the tilt sensor.

js	function <b>get_errorType( )</b>
node.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 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 get_friendlyName( )
nodejs function get_friendlyName( )
php function get_friendlyName( )
cpp string get_friendlyName( )
m -(NSString*) friendlyName
cs string get_friendlyName( )
java String get_friendlyName( )
py def get_friendlyName( )
```

The returned string uses the logical names of the module and of the tilt sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the tilt sensor (for exemple: 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()`  
`tilt->functionDescriptor() tilt->`  
`get_functionDescriptor( )`

YTilt

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

<code>js</code>	<code>function get_functionDescriptor( )</code>
<code>nodejs</code>	<code>function get_functionDescriptor( )</code>
<code>php</code>	<code>function get_functionDescriptor( )</code>
<code>cpp</code>	<code>YFUN_DESCR get_functionDescriptor( )</code>
<code>m</code>	<code>-(YFUN_DESCR) functionDescriptor</code>
<code>pas</code>	<code>function get_functionDescriptor( ): YFUN_DESCR</code>
<code>vb</code>	<code>function get_functionDescriptor( ) As YFUN_DESCR</code>
<code>cs</code>	<code>YFUN_DESCR get_functionDescriptor( )</code>
<code>java</code>	<code>String get_functionDescriptor( )</code>
<code>py</code>	<code>def get_functionDescriptor( )</code>

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>
node.js	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( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def 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()** YTilt  
**tilt→hardwareId() tilt→get\_hardwareId( )**

Returns the unique hardware identifier of the tilt sensor in the form SERIAL.FUNCTIONID.

```
js function get_hardwareId( )
node.js function get_hardwareId( )
php function get_hardwareId( )
cpp string get_hardwareId( )
m -(NSString*) hardwareId
vb function get_hardwareId( ) As String
cs string get_hardwareId( )
java String get_hardwareId( )
py def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the tilt sensor. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the tilt sensor (ex: RELAYL01-123456.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( ): double</b>
vb	function <b>get_highestValue( ) As Double</b>
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YTilt target <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()** YTilt→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 get_logFrequency( )
nodejs function get_logFrequency( )
php   function get_logFrequency( )
cpp   string get_logFrequency( )
m     -(NSString*) logFrequency
pas   function get_logFrequency( ):string
vb    function get_logFrequency( ) As String
cs    string get_logFrequency( )
java  String get_logFrequency( )
py    def get_logFrequency( )
cmd   YTilt target get_logFrequency
```

**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( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	YTilt target <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()** YTilt  
**tilt→lowestValue() tilt→get\_lowestValue( )**

Returns the minimal value observed for the inclination since the device was started.

```
js function get_lowestValue( )
node.js function get_lowestValue( )
php function get_lowestValue( )
cpp double get_lowestValue( )
m -(double) lowestValue
pas function get_lowestValue( ): double
vb function get_lowestValue( ) As Double
cs double get_lowestValue( )
java double get_lowestValue( )
py def get_lowestValue( )
cmd YTilt target get_lowestValue
```

**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 online.

**Returns :**

an instance of YModule

## tilt→get\_module\_async() tilt→module\_async()

YTilt

Gets the `YModule` object for the device on which the function is located (asynchronous version).

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

### Returns :

nothing : the result is provided to the callback.

**tilt→get\_recordedData()**

YTilt

**tilt→recordedData()tilt→get\_recordedData( )**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

<code>js</code>	<code>function get_recordedData( startTime, endTime)</code>
<code>node.js</code>	<code>function get_recordedData( startTime, endTime)</code>
<code>php</code>	<code>function get_recordedData( \$startTime, \$endTime)</code>
<code>cpp</code>	<code>YDataSet get_recordedData( s64 startTime, s64 endTime)</code>
<code>m</code>	<code>-(YDataSet*) recordedData : (s64) startTime : (s64) endTime</code>
<code>pas</code>	<code>function get_recordedData( startTime: int64, endTime: int64): TYDataSet</code>
<code>vb</code>	<code>function get_recordedData( ) As YDataSet</code>
<code>cs</code>	<code>YDataSet get_recordedData( long startTime, long endTime)</code>
<code>java</code>	<code>YDataSet get_recordedData( long startTime, long endTime)</code>
<code>py</code>	<code>def get_recordedData( startTime, endTime)</code>
<code>cmd</code>	<code>YTilt target get_recordedData startTime endTime</code>

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 measure, 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 measure, 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()`  
`tilt→reportFrequency()`  
`tilt→get_reportFrequency( )`

YTilt

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YTilt target get_reportFrequency
```

**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( ): double</b>
vb	function <b>get_resolution( ) As Double</b>
cs	double <b>get_resolution( )</b>
java	double <b>get_resolution( )</b>
py	def <b>get_resolution( )</b>
cmd	YTilt target <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()****YTilt**

Returns the measuring unit for the inclination.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) <b>unit</b>
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	<b>YTilt target 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	<code>function get(userData) </code>
nodejs	<code>function get(userData) </code>
php	<code>function get(userData) </code>
cpp	<code>void * get(userData) </code>
m	<code>-(void*) userData </code>
pas	<code>function get(userData): Tobject </code>
vb	<code>function get(userData) As Object </code>
cs	<code>object get(userData) </code>
java	<code>Object get(userData) </code>
py	<code>def get(userData) </code>

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.

**tilt→isOnline()tilt→isOnline()**

YTilt

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	- <b>(BOOL) 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()**

YTilt

Preloads the tilt sensor cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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()

YTilt

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()`.

<code>js</code>	<code>function nextTilt( )</code>
<code>nodejs</code>	<code>function nextTilt( )</code>
<code>php</code>	<code>function nextTilt( )</code>
<code>cpp</code>	<code>YTilt * nextTilt( )</code>
<code>m</code>	<code>-(YTilt*) nextTilt</code>
<code>pas</code>	<code>function nextTilt( ): TYTilt</code>
<code>vb</code>	<code>function nextTilt( ) As YTilt</code>
<code>cs</code>	<code>YTilt nextTilt( )</code>
<code>java</code>	<code>YTilt nextTilt( )</code>
<code>py</code>	<code>def nextTilt( )</code>

**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 registerTimedReportCallback( callback )
node.js function registerTimedReportCallback( callback )
php  function registerTimedReportCallback( $callback )
cpp   int registerTimedReportCallback( YTiltTimedReportCallback callback )
m    -(int) registerTimedReportCallback : (YTiltTimedReportCallback) callback
pas   function registerTimedReportCallback( callback: YTiltTimedReportCallback): LongInt
vb    function registerTimedReportCallback( ) As Integer
cs    int registerTimedReportCallback( TimedReportCallback callback )
java  int registerTimedReportCallback( TimedReportCallback callback )
py    def registerTimedReportCallback( callback )
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

**Parameters :**

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and 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 registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php  function registerValueCallback( $callback)
cpp   int registerValueCallback( YTiltValueCallback callback)
m    -(int) registerValueCallback : (YTiltValueCallback) callback
pas   function registerValueCallback( callback: TYTiltValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs   int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
py    def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

### Parameters :

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

**tilt→set\_highestValue()  
tilt→setHighestValue()tilt→set\_highestValue( )**

YTilt

Changes the recorded maximal value observed.

```
js function set_highestValue( newval)
node.js function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YTilt target set_highestValue newval
```

**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() tilt→setLogFrequency() tilt→ set\_logFrequency( )

YTilt

Changes the datalogger recording frequency for this function.

js	function set_logFrequency( newval)
node.js	function set_logFrequency( newval)
php	function set_logFrequency( \$newval)
cpp	int set_logFrequency( const string& newval)
m	-(int) setLogFrequency : (NSString*) newval
pas	function set_logFrequency( newval: string): integer
vb	function set_logFrequency( ByVal newval As String) As Integer
cs	int set_logFrequency( string newval)
java	int set_logFrequency( String newval)
py	def set_logFrequency( newval)
cmd	YTilt target set_logFrequency 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 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 set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YTilt target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

**Parameters :**

**newval** a string corresponding to the logical name of the 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 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 YTilt target set_lowestValue newval
```

**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( )`

YTilt

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   YTilt 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.

**tilt→set\_resolution()**

YTilt

**tilt→setResolution()tilt→set\_resolution( )**

Changes the resolution of the measured physical values.

js	<code>function set_resolution( newval)</code>
nodejs	<code>function set_resolution( newval)</code>
php	<code>function set_resolution( \$newval)</code>
cpp	<code>int set_resolution( double newval)</code>
m	<code>-(int) setResolution : (double) newval</code>
pas	<code>function set_resolution( newval: double): integer</code>
vb	<code>function set_resolution( ByVal newval As Double) As Integer</code>
cs	<code>int set_resolution( double newval)</code>
java	<code>int set_resolution( double newval)</code>
py	<code>def set_resolution( newval)</code>
cmd	<code>YTilt target set_resolution newval</code>

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()**

YTilt

Stores a user context provided as argument in the userData attribute of the function.

```
js   function set(userData) {  
node.js function set(userData) {  
php  function set(userData) {  
cpp   void set(userData) {  
m     -(void) setUserData : (void*) userData  
pas   procedure set(userData: Tobject);  
vb    procedure set(userData: ByVal userData As Object);  
cs    void set(userData: object);  
java  void set(userData: Object);  
py    def set(userData: object);
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

**Parameters :**

**userData** 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');
cpp	#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()yFindVoc( )**

Retrieves a Volatile Organic Compound sensor for a given identifier.

<code>js</code>	<code>function yFindVoc( func)</code>
<code>node.js</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()yFirstVoc( )**

Starts the enumeration of Volatile Organic Compound sensors currently accessible.

**js** function **yFirstVoc( )****nodejs** function **FirstVoc( )****php** function **yFirstVoc( )****cpp** YVoc\* **yFirstVoc( )****m** YVoc\* **yFirstVoc( )****pas** function **yFirstVoc( ): TYVoc****vb** function **yFirstVoc( ) As YVoc****cs** YVoc **FirstVoc( )****java** YVoc **FirstVoc( )****py** def **FirstVoc( )**

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)
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  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()****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	- <b>(NSString*) describe</b>
pas	function <b>describe( )</b> : string
vb	function <b>describe( )</b> As String
cs	string <b>describe( )</b>
java	<b>String describe( )</b>
py	<b>def 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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**voc→get\_advertisedValue()**  
**voc→advertisedValue()voc→**  
**get\_advertisedValue( )**

**YVoc**

Returns the current value of the Volatile Organic Compound sensor (no more than 6 characters).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YVoc target 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()**  
**voc→currentRawValue()voc→**  
**get\_currentRawValue( )**

**YVoc**

Returns the unrounded and uncalibrated raw value returned by the sensor.

**js** function **get\_currentRawValue( )**  
**nodejs** function **get\_currentRawValue( )**  
**php** function **get\_currentRawValue( )**  
**cpp** double **get\_currentRawValue( )**  
**m** -(double) currentRawValue  
**pas** function **get\_currentRawValue( ): double**  
**vb** function **get\_currentRawValue( ) As Double**  
**cs** double **get\_currentRawValue( )**  
**java** double **get\_currentRawValue( )**  
**py** def **get\_currentRawValue( )**  
**cmd** YVoc **target get\_currentRawValue**

**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>
node.js	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	<b>YVoc target 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>
node.js	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	- <b>(NSString*) errorMessage</b>
pas	function <b>get_errorMessage( ): string</b>
vb	function <b>get_errorMessage( ) As String</b>
cs	string <b>get_errorMessage( )</b>
java	<b>String getErrorMessage( )</b>
py	<b>def getErrorMessage( )</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 get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the 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()**  
**voc→functionDescriptor()voc→**  
**get\_functionDescriptor( )**

**YVoc**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor( )</b>
node.js	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	<b>String get_functionDescriptor( )</b>
py	<b>def 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()**  
**voc→functionId()voc→get\_functionId()****YVoc**

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( ) As String</b>
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 get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the Volatile Organic Compound sensor. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the Volatile Organic Compound sensor (ex: RELAYL01-123456.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>
node.js	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	<b>YVoc target 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( ): string</b>
vb	function <b>get_logFrequency( ) As String</b>
cs	string <b>get_logFrequency( )</b>
java	String <b>get_logFrequency( )</b>
py	def <b>get_logFrequency( )</b>
cmd	YVoc <b>target 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>
node.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( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	<b>YVoc target 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( ): double</b>
vb	function <b>get_lowestValue( ) As Double</b>
cs	double <b>get_lowestValue( )</b>
java	double <b>get_lowestValue( )</b>
py	def <b>get_lowestValue( )</b>
cmd	<b>YVoc target 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()**  
**voc->module()voc->get\_module()**

**YVoc**

Gets the `YModule` object for the device on which the function is located.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

If the function cannot be located on any module, the returned instance of `YModule` is not shown as online.

**Returns :**

an instance of `YModule`

**voc→get\_module\_async()****YVoc****voc→module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

```
js   function get_module_async( callback, context )
nodejs function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**voc→get\_recordedData()****YVoc****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( startTime, endTime)</b>
node.js	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	YDataSet <b>get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime : (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
java	YDataSet <b>get_recordedData( long startTime, long endTime)</b>
py	<b>def get_recordedData( startTime, endTime)</b>
cmd	<b>YVoc target get_recordedData startTime 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 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()**  
**voc→resolution()voc→get\_resolution()****YVoc**

Returns the resolution of the measured values.

js	function <b>get_resolution( )</b>
node.js	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	<b>YVoc target 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*) <b>unit</b>
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 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)**  
**voc→userData()voc→get(userData()****YVoc**

Returns the value of the userData attribute, as previously stored using method set(userData).

js	function get(userData)
node.js	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): TObject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

**Returns :**  
the object stored previously by the caller.

**voc→isOnline()voc→isOnline( )****YVoc**

Checks if the Volatile Organic Compound sensor is currently reachable, without raising any error.

js	function <b>isOnline()</b>
node.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 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()****YVoc**

Preloads the Volatile Organic Compound sensor cache with a specified validity duration.

js	function <b>load( msValidity)</b>
node.js	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	-(YRETCODE) <b>load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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)
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   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()****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	<b>YVoc * nextVoc()</b>
m	<b>-(YVoc*) nextVoc</b>
pas	function <b>nextVoc()</b> : TYVoc
vb	function <b>nextVoc()</b> As YVoc
cs	<b>YVoc nextVoc()</b>
java	<b>YVoc nextVoc()</b>
py	<b>def 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 registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YVocTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YVocTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYVocTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YVocValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YVocValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYVocValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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 set_highestValue( newval)
nodejs function set_highestValue( newval)
php  function set_highestValue( $newval)
cpp   int set_highestValue( double newval)
m    -(int) setHighestValue : (double) newval
pas   function set_highestValue( newval: double): integer
vb    function set_highestValue( ByVal newval As Double) As Integer
cs    int set_highestValue( double newval)
java  int set_highestValue( double newval)
py    def set_highestValue( newval)
cmd   YVoc target set_highestValue newval
```

**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()**  
**voc→setLogFrequency()****voc→set\_logFrequency( )****YVoc**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency( newval)</b>
node.js	function <b>set_logFrequency( newval)</b>
php	function <b>set_logFrequency( \$newval)</b>
cpp	int <b>set_logFrequency( const string&amp; newval)</b>
m	- (int) <b>setLogFrequency : (NSString*) newval</b>
pas	function <b>set_logFrequency( newval: string): integer</b>
vb	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
cs	int <b>set_logFrequency( string newval)</b>
java	int <b>set_logFrequency( String newval)</b>
py	def <b>set_logFrequency( newval)</b>
cmd	<b>YVoc 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.

<b>js</b>	<code>function set_logicalName( newval)</code>
<b>nodejs</b>	<code>function set_logicalName( newval)</code>
<b>php</b>	<code>function set_logicalName( \$newval)</code>
<b>cpp</b>	<code>int set_logicalName( const string&amp; newval)</code>
<b>m</b>	<code>-(int) setLogicalName : (NSString*) newval</code>
<b>pas</b>	<code>function set_logicalName( newval: string): integer</code>
<b>vb</b>	<code>function set_logicalName( ByVal newval As String) As Integer</code>
<b>cs</b>	<code>int set_logicalName( string newval)</code>
<b>java</b>	<code>int set_logicalName( String newval)</code>
<b>py</b>	<code>def set_logicalName( newval)</code>
<b>cmd</b>	<code>YVoc target set_logicalName newval</code>

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()**  
**voc→setLowestValue()voc→set\_lowestValue( )****YVoc**

Changes the recorded minimal value observed for the estimated VOC concentration.

```
js function set_lowestValue( newval)
node.js 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 YVoc target set_lowestValue newval
```

**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()**  
**voc→setReportFrequency()** **voc→**  
**set\_reportFrequency( )**

YVoc

---

Changes the timed value notification frequency for this function.

<b>js</b>	function <b>set_reportFrequency( newval)</b>
<b>node.js</b>	function <b>set_reportFrequency( newval)</b>
<b>php</b>	function <b>set_reportFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_reportFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setReportFrequency : (NSString*) newval</b>
<b>pas</b>	function <b>set_reportFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_reportFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_reportFrequency( string newval)</b>
<b>java</b>	int <b>set_reportFrequency( String newval)</b>
<b>py</b>	def <b>set_reportFrequency( newval)</b>
<b>cmd</b>	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()**  
**voc->set\_resolution( )****YVoc**

Changes the resolution of the measured physical values.

js	function <b>set_resolution( newval)</b>
node.js	function <b>set_resolution( newval)</b>
php	function <b>set_resolution( \$newval)</b>
cpp	int <b>set_resolution( double newval)</b>
m	-{int) setResolution : (double) <b>newval</b>
pas	function <b>set_resolution( newval: double): integer</b>
vb	function <b>set_resolution( ByVal newval As Double) As Integer</b>
cs	int <b>set_resolution( double newval)</b>
java	int <b>set_resolution( double newval)</b>
py	def <b>set_resolution( newval)</b>
cmd	<b>YVoc 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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b> ( void* <b>data</b> )
m	-(void) <b>set(userData : (void*) data</b>
pas	procedure <b>set(userData : Tobject)</b>
vb	procedure <b>set(userData : ByVal data As Object)</b>
cs	void <b>set(userData : object data)</b>
java	void <b>set(userData : Object data)</b>
py	def <b>set(userData : 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');
cpp	#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.

<b>voltage→get_highestValue()</b>	Returns the maximal value observed for the voltage.
<b>voltage→get_logFrequency()</b>	Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.
<b>voltage→get_logicalName()</b>	Returns the logical name of the voltage sensor.
<b>voltage→get_lowestValue()</b>	Returns the minimal value observed for the voltage.
<b>voltage→get_module()</b>	Gets the YModule object for the device on which the function is located.
<b>voltage→get_module_async(callback, context)</b>	Gets the YModule object for the device on which the function is located (asynchronous version).
<b>voltage→get_recordedData(startTime, endTime)</b>	Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.
<b>voltage→get_reportFrequency()</b>	Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.
<b>voltage→get_resolution()</b>	Returns the resolution of the measured values.
<b>voltage→get_unit()</b>	Returns the measuring unit for the voltage.
<b>voltage→get(userData)</b>	Returns the value of the userData attribute, as previously stored using method set(userData).
<b>voltage→isOnline()</b>	Checks if the voltage sensor is currently reachable, without raising any error.
<b>voltage→isOnline_async(callback, context)</b>	Checks if the voltage sensor is currently reachable, without raising any error (asynchronous version).
<b>voltage→load(msValidity)</b>	Preloads the voltage sensor cache with a specified validity duration.
<b>voltage→loadCalibrationPoints(rawValues, refValues)</b>	Retrieves error correction data points previously entered using the method calibrateFromPoints.
<b>voltage→load_async(msValidity, callback, context)</b>	Preloads the voltage sensor cache with a specified validity duration (asynchronous version).
<b>voltage→nextVoltage()</b>	Continues the enumeration of voltage sensors started using yFirstVoltage( ).
<b>voltage→registerTimedReportCallback(callback)</b>	Registers the callback function that is invoked on every periodic timed notification.
<b>voltage→registerValueCallback(callback)</b>	Registers the callback function that is invoked on every change of advertised value.
<b>voltage→set_highestValue(newval)</b>	Changes the recorded maximal value observed pour the voltage.
<b>voltage→set_logFrequency(newval)</b>	Changes the datalogger recording frequency for this function.
<b>voltage→set_logicalName(newval)</b>	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()yFindVoltage( )

YVoltage

Retrieves a voltage sensor for a given identifier.

```
js function yFindVoltage( func)
node.js function FindVoltage( func)
php function yFindVoltage( $func)
cpp YVoltage* yFindVoltage( const string& func)
m YVoltage* yFindVoltage( NSString* func)
pas function yFindVoltage( func: string): TYVoltage
vb function yFindVoltage( ByVal func As String) As YVoltage
cs YVoltage FindVoltage( string func)
java YVoltage FindVoltage( String func)
py def FindVoltage( 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 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()****YVoltage****yFirstVoltage()yFirstVoltage( )**

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)=RELAYL01-123456.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 voltage sensor (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**voltage→get\_advertisedValue()**  
**voltage→advertisedValue()**  
**voltage→get\_advertisedValue( )**

**YVoltage**

Returns the current value of the voltage sensor (no more than 6 characters).

**js** function **get\_advertisedValue( )**  
**nodejs** function **get\_advertisedValue( )**  
**php** function **get\_advertisedValue( )**  
**cpp** string **get\_advertisedValue( )**  
**m** -(NSString\*) advertisedValue  
**pas** function **get\_advertisedValue( )**: string  
**vb** function **get\_advertisedValue( )** As String  
**cs** string **get\_advertisedValue( )**  
**java** String **get\_advertisedValue( )**  
**py** def **get\_advertisedValue( )**  
**cmd** YVoltage **target get\_advertisedValue**

**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()**  
**voltage→currentRawValue()voltage→**  
**get\_currentRawValue( )**

**YVoltage**

Returns the uncalibrated, unrounded raw value returned by the sensor.

```
js function get_currentRawValue( )
nodejs function get_currentRawValue( )
php function get_currentRawValue( )
cpp double get_currentRawValue( )
m -(double) currentRawValue
pas function get_currentRawValue( ): double
vb function get_currentRawValue( ) As Double
cs double get_currentRawValue( )
java double get_currentRawValue( )
py def get_currentRawValue( )
cmd YVoltage target get_currentRawValue
```

**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 get_currentValue( )
nodejs function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YVoltage target get_currentValue
```

**Returns :**

a floating point number corresponding to the current measure for the voltage

On failure, throws an exception or returns Y\_CURRENTVALUE\_INVALID.

**voltage→getErrorMessage()**  
**voltage→errorMessage()voltage→**  
**getErrorMessage( )**

**YVoltage**

Returns the error message of the latest error with the voltage sensor.

js	function getErrorMessage( )
node.js	function getErrorMessage( )
php	function getErrorMessage( )
cpp	string getErrorMessage( )
m	-(NSString*) errorMessage
pas	function getErrorMessage( ): string
vb	function getErrorMessage( ) As String
cs	string getErrorMessage( )
java	String getErrorMessage( )
py	def getErrorMessage( )

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>
node.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 the voltage sensor object

**voltage→get\_friendlyName()**  
**voltage→friendlyName()voltage→**  
**get\_friendlyName( )**

**YVoltage**

Returns a global identifier of the voltage sensor in the format MODULE\_NAME . FUNCTION\_NAME.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the voltage sensor if they are defined, otherwise the serial number of the module and the hardware identifier of the voltage sensor (for exemple: 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()**  
**voltage→functionDescriptor()voltage→**  
**get\_functionDescriptor( )**

**YVoltage**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

<code>js</code>	function <b>get_functionDescriptor( )</b>
<code>nodejs</code>	function <b>get_functionDescriptor( )</b>
<code>php</code>	function <b>get_functionDescriptor( )</b>
<code>cpp</code>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<code>m</code>	-(YFUN_DESCR) <b>functionDescriptor</b>
<code>pas</code>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<code>vb</code>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<code>cs</code>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<code>java</code>	String <b>get_functionDescriptor( )</b>
<code>py</code>	<b>def 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>
node.js	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( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def 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 get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the voltage sensor. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the voltage sensor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**voltage→get\_highestValue()**  
**voltage→highestValue()voltage→**  
**get\_highestValue()**

YVoltage

Returns the maximal value observed for the voltage.

js	function <b>get_highestValue( )</b>
node.js	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 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()**  
**voltage→logFrequency()voltage→**  
**get\_logFrequency( )**

**YVoltage**

Returns the datalogger recording frequency for this function, or "OFF" when measures are not stored in the data logger flash memory.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ):string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YVoltage target get_logFrequency
```

**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()**  
**voltage→logicalName()voltage→**  
**get\_logicalName( )**

**YVoltage**

Returns the logical name of the voltage sensor.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	<b>YVoltage target 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()**  
**voltage→lowestValue()** **voltage→get\_lowestValue()**

**YVoltage**

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	<b>YVoltage target 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 online.

**Returns :**

an instance of YModule

**voltage→get\_module\_async()**  
**voltage→module\_async()****YVoltage**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**voltage→get\_recordedData()**  
**voltage→recordedData()voltage→**  
**get\_recordedData()**

**YVoltage**

Retrieves a DataSet object holding historical data for this sensor, for a specified time interval.

js	function <b>get_recordedData( startTime, endTime)</b>
node.js	function <b>get_recordedData( startTime, endTime)</b>
php	function <b>get_recordedData( \$startTime, \$endTime)</b>
cpp	<b>YDataSet get_recordedData( s64 startTime, s64 endTime)</b>
m	- <b>(YDataSet*) recordedData : (s64) startTime</b> <b>: (s64) endTime</b>
pas	function <b>get_recordedData( startTime: int64, endTime: int64): TYDataSet</b>
vb	function <b>get_recordedData( ) As YDataSet</b>
cs	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
java	<b>YDataSet get_recordedData( long startTime, long endTime)</b>
py	<b>def get_recordedData( startTime, endTime)</b>
cmd	<b>YVoltage target get_recordedData startTime 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 measure, 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 measure, 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()**  
**voltage→reportFrequency()voltage→**  
**get\_reportFrequency( )**

**YVoltage**

Returns the timed value notification frequency, or "OFF" if timed value notifications are disabled for this function.

<b>js</b>	function <b>get_reportFrequency( )</b>
<b>nodejs</b>	function <b>get_reportFrequency( )</b>
<b>php</b>	function <b>get_reportFrequency( )</b>
<b>cpp</b>	string <b>get_reportFrequency( )</b>
<b>m</b>	-(NSString*) reportFrequency
<b>pas</b>	function <b>get_reportFrequency( ): string</b>
<b>vb</b>	function <b>get_reportFrequency( ) As String</b>
<b>cs</b>	string <b>get_reportFrequency( )</b>
<b>java</b>	String <b>get_reportFrequency( )</b>
<b>py</b>	def <b>get_reportFrequency( )</b>
<b>cmd</b>	<b>YVoltage target 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( ): double</b>
vb	function <b>get_resolution( ) As Double</b>
cs	double <b>get_resolution( )</b>
java	double <b>get_resolution( )</b>
py	def <b>get_resolution( )</b>
cmd	<b>YVoltage target 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>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) <b>unit</b>
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	<b>YVoltage target 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	<code>function get(userData( )</code>
nodejs	<code>function get(userData( )</code>
php	<code>function get(userData( )</code>
cpp	<code>void * get(userData( )</code>
m	<code>-(void*) userData</code>
pas	<code>function get(userData( ): Tobject</code>
vb	<code>function get(userData( ) As Object</code>
cs	<code>object get(userData( )</code>
java	<code>Object get(userData( )</code>
py	<code>def get(userData( )</code>

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()****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	- <b>(BOOL) 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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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()****YVoltage**

Continues the enumeration of voltage sensors started using `yFirstVoltage()`.

js	function <b>nextVoltage()</b>
node.js	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.

<b>js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>node.js</b>	function registerTimedReportCallback( <b>callback</b> )
<b>php</b>	function registerTimedReportCallback( <b>\$callback</b> )
<b>cpp</b>	int registerTimedReportCallback( YVoltageTimedReportCallback <b>callback</b> )
<b>m</b>	- (int) registerTimedReportCallback : (YVoltageTimedReportCallback) <b>callback</b>
<b>pas</b>	function registerTimedReportCallback( <b>callback</b> : TYVoltageTimedReportCallback): LongInt
<b>vb</b>	function registerTimedReportCallback( ) As Integer
<b>cs</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>java</b>	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
<b>py</b>	def registerTimedReportCallback( <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 registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php  function registerValueCallback( $callback)
cpp   int registerValueCallback( YVoltageValueCallback callback)
m    -(int) registerValueCallback : (YVoltageValueCallback) callback
pas   function registerValueCallback( callback: TYVoltageValueCallback): LongInt
vb    function registerValueCallback( ) As Integer
cs   int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
py    def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

**Parameters :**

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

**voltage→set\_highestValue()****YVoltage****voltage→setHighestValue()voltage→  
set\_highestValue( )**

Changes the recorded maximal value observed pour the voltage.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YVoltage target set_highestValue newval
```

**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()**  
**voltage→setLogFrequency()voltage→**  
**set\_logFrequency( )**

**YVoltage**

Changes the datalogger recording frequency for this function.

js	function <b>set_logFrequency( newval)</b>
node.js	function <b>set_logFrequency( newval)</b>
php	function <b>set_logFrequency( \$newval)</b>
cpp	int <b>set_logFrequency( const string&amp; newval)</b>
m	-(int) <b>setLogFrequency : (NSString*) newval</b>
pas	function <b>set_logFrequency( newval: string): integer</b>
vb	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
cs	int <b>set_logFrequency( string newval)</b>
java	int <b>set_logFrequency( String newval)</b>
py	def <b>set_logFrequency( newval)</b>
cmd	YVoltage <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.

**voltage→set\_logicalName()**  
**voltage→setLogicalName()** voltage→  
**set\_logicalName( )**

**YVoltage**

Changes the logical name of the voltage sensor.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YVoltage target set_logicalName newval
```

You can use `yCheckLogicalName( )` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash( )` method of the module if the modification must be kept.

**Parameters :**

**newval** a string corresponding to the logical name of the voltage sensor.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

**voltage→set\_lowestValue()**  
**voltage→setLowestValue()voltage→**  
**set\_lowestValue( )**

**YVoltage**

Changes the recorded minimal value observed pour the voltage.

<b>js</b>	function <b>set_lowestValue( newval)</b>
<b>nodejs</b>	function <b>set_lowestValue( newval)</b>
<b>php</b>	function <b>set_lowestValue( \$newval)</b>
<b>cpp</b>	int <b>set_lowestValue( double newval)</b>
<b>m</b>	-(int) <b>setLowestValue : (double) newval</b>
<b>pas</b>	function <b>set_lowestValue( newval: double): integer</b>
<b>vb</b>	function <b>set_lowestValue( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_lowestValue( double newval)</b>
<b>java</b>	int <b>set_lowestValue( double newval)</b>
<b>py</b>	def <b>set_lowestValue( newval)</b>
<b>cmd</b>	<b>YVoltage target set_lowestValue 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()**  
**voltage→setReportFrequency()** **voltage→set\_reportFrequency( )**

**YVoltage**

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   YVoltage 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.

**voltage→set\_resolution()**  
**voltage→setResolution()****voltage→**  
**set\_resolution( )**

**YVoltage**

Changes the resolution of the measured values.

js	function <b>set_resolution( newval)</b>
node.js	function <b>set_resolution( newval)</b>
php	function <b>set_resolution( \$newval)</b>
cpp	int <b>set_resolution( double newval)</b>
m	-(int) <b>setResolution : (double) newval</b>
pas	function <b>set_resolution( newval: double): integer</b>
vb	function <b>set_resolution( ByVal newval As Double) As Integer</b>
cs	int <b>set_resolution( double newval)</b>
java	int <b>set_resolution( double newval)</b>
py	def <b>set_resolution( newval)</b>
cmd	<b>YVoltage 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>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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');
cpp	#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 YModule object for the device on which the function is located (asynchronous version).

**vsouce→get\_overCurrent()**

Returns true if the appliance connected to the device is too greedy .

**vsouce→get\_overHeat()**

Returns TRUE if the module is overheating.

**vsouce→get\_overLoad()**

Returns true if the device is not able to maintain the requested voltage output .

**vsouce→get\_regulationFailure()**

Returns true if the voltage output is too high regarding the requested voltage .

**vsouce→get\_unit()**

Returns the measuring unit for the voltage.

**vsouce→get(userData)**

Returns the value of the userData attribute, as previously stored using method set(userData).

**vsouce→get\_voltage()**

Returns the voltage output command (mV)

**vsouce→isOnline()**

Checks if the function is currently reachable, without raising any error.

**vsouce→isOnline\_async(callback, context)**

Checks if the function is currently reachable, without raising any error (asynchronous version).

**vsouce→load(msValidity)**

Preloads the function cache with a specified validity duration.

**vsouce→load\_async(msValidity, callback, context)**

Preloads the function cache with a specified validity duration (asynchronous version).

**vsouce→nextVSource()**

Continues the enumeration of voltage sources started using yFirstVSource( ).

**vsouce→pulse(voltage, ms\_duration)**

Sets device output to a specific volatage, for a specified duration, then brings it automatically to 0V.

**vsouce→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**vsouce→set\_logicalName(newval)**

Changes the logical name of the voltage source.

**vsouce→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**vsouce→set\_voltage(newval)**

Tunes the device output voltage (milliVolts).

**vsouce→voltageMove(target, ms\_duration)**

Performs a smooth move at constant speed toward a given value.

**vsouce→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## yFindVSource() — YVSource.FindVSource()`yFindVSource( )`

Retrieves a voltage source for a given identifier.

```
js function yFindVSource( func)
php function yFindVSource( $func)
cpp YVSource* yFindVSource( const string& func)
m YVSource* yFindVSource( NSString* func)
pas function yFindVSource( func: string): TYVSource
vb function yFindVSource( ByVal func As String) As YVSource
cs YVSource FindVSource( string func)
java YVSource FindVSource( String func)
py def FindVSource( 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 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()yFirstVSource( )**

Starts the enumeration of voltage sources currently accessible.

```
js   function yFirstVSource( )
php  function yFirstVSource( )
cpp  YVSource* yFirstVSource( )
m    YVSource* yFirstVSource( )
pas   function yFirstVSource( ): TYVSource
vb    function yFirstVSource( ) As YVSource
cs    YVSource FirstVSource( )
java  YVSource FirstVSource( )
py    def FirstVSource( )
```

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()****YVSource**

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 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)=RELAYL01-123456.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 function (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**vsource→get\_advertisedValue()**  
**vsource→advertisedValue()vsource→**  
**get\_advertisedValue( )**

**YVSource**

Returns the current value of the voltage source (no more than 6 characters).

**js** function **get\_advertisedValue( )**  
**php** function **get\_advertisedValue( )**  
**cpp** string **get\_advertisedValue( )**  
**m** -(NSString\*) **advertisedValue**  
**pas** function **get\_advertisedValue( )**: string  
**vb** function **get\_advertisedValue( )** As String  
**cs** string **get\_advertisedValue( )**  
**java** String **get\_advertisedValue( )**  
**py** def **get\_advertisedValue( )**  
**cmd** YVSource **target get\_advertisedValue**

**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**.

**vsource→getErrorMessage()**  
**vsource→errorMessage()vsource→**  
**getErrorMessage( )**

**YVSource**

Returns the error message of the latest error with this function.

```
js  function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ):string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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 get_errorType( )
php  function get_errorType( )
cpp  YRETCODE get_errorType( )
pas   function get_errorType( ): YRETCODE
vb    function get_errorType( ) As YRETCODE
cs    YRETCODE get_errorType( )
java  int get_errorType( )
py    def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a number corresponding to the code of the latest error that occurred while using this function object

**vsOURCE→get\_extPowerFailure()**  
**vsOURCE→extPowerFailure()vsOURCE→**  
**get\_extPowerFailure( )**

**YVSource**

Returns true if external power supply voltage is too low.

```
js function get_extPowerFailure( )
php function get_extPowerFailure( )
cpp Y_EXTPOWERFAILURE_enum get_extPowerFailure( )
m -(Y_EXTPOWERFAILURE_enum) extPowerFailure
pas function get_extPowerFailure( ): Integer
vb function get_extPowerFailure( ) As Integer
cs int get_extPowerFailure( )
java int get_extPowerFailure( )
py def get_extPowerFailure( )
cmd YVSource target get_extPowerFailure
```

**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 get_failure( )
php function get_failure( )
cpp Y_FAILURE_enum get_failure( )
m -(Y_FAILURE_enum) failure
pas function get_failure( ): Integer
vb function get_failure( ) As Integer
cs int get_failure( )
java int get_failure( )
py def get_failure( )
cmd YVSource target get_failure
```

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.

**vsOURCE→get\_friendlyName()**  
**vsOURCE→friendlyName()vsOURCE→**  
**get\_friendlyName( )**

**YVSource**

Returns a global identifier of the function in the format MODULE\_NAME . FUNCTION\_NAME.

**js** function **get\_friendlyName( )**  
**php** function **get\_friendlyName( )**  
**cpp** virtual string **get\_friendlyName( )**  
**m** -(NSString\*) friendlyName  
**cs** override string **get\_friendlyName( )**  
**java** String **get\_friendlyName( )**

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.

**vsource→get\_functionDescriptor()**  
**vsource→functionDescriptor()vsource→**  
**get\_vsourceDescriptor( )**

**YVSource**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

```
js  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.

**vsource→get\_functionId()****YVSource****vsource→functionId()vsource→get\_vsourceId( )**

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*) <b>functionId</b>
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()**  
**vsource→hardwareId()vsource→**  
**get\_hardwareId( )**

**YVSource**

Returns the unique hardware identifier of the function in the form SERIAL.FUNCTIONID.

```
js  function get_hardwareId( )
php function get_hardwareId( )
cpp string get_hardwareId( )
m -(NSString*) hardwareId
vb function get_hardwareId( ) As String
cs string get_hardwareId( )
java String get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the function. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the function (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

---

<b>vsOURCE→get_logicalName()</b>	<b>YVSource</b>
<b>vsOURCE→logicalName()vsOURCE→get_logicalName( )</b>	

---

Returns the logical name of the voltage source.

```
js function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YVSource target get_logicalName
```

**Returns :**

a string corresponding to the logical name of the voltage source

On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

**vsource→get\_module()****YVSource****vsource→module()vsource→get\_module()**

Gets the YModule object for the device on which the function is located.

```
js   function get_module( )
php  function get_module( )
cpp  YModule * get_module( )
m    -(YModule*) module
pas   function get_module( ): TYModule
vb    function get_module( ) As YModule
cs    YModule get_module( )
java  YModule get_module( )
py    def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

**Returns :**

an instance of YModule

**vsources→get\_module\_async()**  
**vsources→module\_async()****YVSource**

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 online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**vsource→get\_overCurrent()**  
**vsource→overCurrent()vsource→**  
**get\_overCurrent( )**

**YVSource**

Returns true if the appliance connected to the device is too greedy .

```
js function get_overCurrent( )
php function get_overCurrent( )
cpp Y_OVERCURRENT_enum get_overCurrent( )
m -(Y_OVERCURRENT_enum) overCurrent
pas function get_overCurrent( ): Integer
vb function get_overCurrent( ) As Integer
cs int get_overCurrent( )
java int get_overCurrent( )
py def get_overCurrent( )
cmd YVSource target get_overCurrent
```

**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 get_overHeat( )
php function get_overHeat( )
cpp Y_OVERHEAT_enum get_overHeat( )
m -(Y_OVERHEAT_enum) overHeat
pas function get_overHeat( ): Integer
vb function get_overHeat( ) As Integer
cs int get_overHeat( )
java int get_overHeat( )
py def get_overHeat( )
cmd YVSource target get_overHeat
```

**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 maintain the requested voltage output .

```
js   function get_overLoad( )
php  function get_overLoad( )
cpp  Y_OVERLOAD_enum get_overLoad( )
m    -(Y_OVERLOAD_enum) overLoad
pas   function get_overLoad( ): Integer
vb    function get_overLoad( ) As Integer
cs    int get_overLoad( )
java  int get_overLoad( )
py    def get_overLoad( )
cmd   YVSource target get_overLoad
```

**Returns :**

either Y\_OVERLOAD\_FALSE or Y\_OVERLOAD\_TRUE, according to true if the device is not able to maintain the requested voltage output

On failure, throws an exception or returns Y\_OVERLOAD\_INVALID.

**vsOURCE→get\_regulationFailure()**  
**vsOURCE→regulationFailure()vsOURCE→**  
**get\_regulationFailure( )**

**YVSource**

Returns true if the voltage output is too high regarding the requested voltage .

```
js function get_regulationFailure( )
php function get_regulationFailure( )
cpp Y_REGULATIONFAILURE_enum get_regulationFailure( )
m -(Y_REGULATIONFAILURE_enum) regulationFailure
pas function get_regulationFailure( ): Integer
vb function get_regulationFailure( ) As Integer
cs int get_regulationFailure( )
java int get_regulationFailure( )
py def get_regulationFailure( )
cmd YVSource target get_regulationFailure
```

**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 get_unit( )
php  function get_unit( )
cpp  string get_unit( )
m    -(NSString*) unit
pas  function get_unit( ): string
vb   function get_unit( ) As String
cs   string get_unit( )
java String get_unit( )
py   def get_unit( )
cmd  YVSource target get_unit
```

**Returns :**

a string corresponding to the measuring unit for the voltage

On failure, throws an exception or returns Y\_UNIT\_INVALID.

**vsouce→get(userData)****YVSource****vsouce→userData()vsouce→get(userData()**

Returns the value of the userData attribute, as previously stored using method set(userData).

js	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): Tobject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

**Returns :**

the object stored previously by the caller.

**vsource→get\_voltage()****YVSource****vsource→voltage()vsource→get\_voltage()**

Returns the voltage output command (mV)

```
js   function get_voltage( )
php  function get_voltage( )
cpp  int get_voltage( )
m    -(int) voltage
pas  function get_voltage( ): LongInt
vb   function get_voltage( ) As Integer
cs   int get_voltage( )
java int get_voltage( )
py   def get_voltage( )
```

**Returns :**

an integer corresponding to the voltage output command (mV)

On failure, throws an exception or returns Y\_VOLTAGE\_INVALID.

**vsource→isOnline()|vsource→isOnline()****YVSource**

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.

**vsources->load()****YVSource**

Preloads the function cache with a specified validity duration.

js	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	<b>YRETCODE load( int msValidity)</b>
m	<b>-(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	<b>YRETCODE load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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.

**vsources->nextVSource()****YVSource**

Continues the enumeration of voltage sources started using `yFirstVSource()`.

<code>js</code>	<code>function nextVSource()</code>
<code>php</code>	<code>function nextVSource()</code>
<code>cpp</code>	<code>YVSource * nextVSource()</code>
<code>m</code>	<code>-(YVSource*) nextVSource</code>
<code>pas</code>	<code>function nextVSource(): TYVSource</code>
<code>vb</code>	<code>function nextVSource() As YVSource</code>
<code>cs</code>	<code>YVSource nextVSource()</code>
<code>java</code>	<code>YVSource nextVSource()</code>
<code>py</code>	<code>def nextVSource()</code>

**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()****YVSource**

Sets device output to a specific voltage, for a specified duration, then brings it automatically to 0V.

<b>js</b>	<code>function pulse( voltage, ms_duration)</code>
<b>php</b>	<code>function pulse( \$voltage, \$ms_duration)</code>
<b>cpp</b>	<code>int pulse( int voltage, int ms_duration)</code>
<b>m</b>	<code>-(int) pulse : (int) voltage : (int) ms_duration</code>
<b>pas</b>	<code>function pulse( voltage: integer, ms_duration: integer): integer</code>
<b>vb</b>	<code>function pulse( ByVal voltage As Integer, ByVal ms_duration As Integer) As Integer</code>
<b>cs</b>	<code>int pulse( int voltage, int ms_duration)</code>
<b>java</b>	<code>int pulse( int voltage, int ms_duration)</code>
<b>py</b>	<code>def pulse( voltage, ms_duration)</code>
<b>cmd</b>	<code>YVSource target pulse voltage ms_duration</code>

**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.

**vsource→registerValueCallback()**  
**vsource→registerValueCallback( )**

**YVSource**

Registers the callback function that is invoked on every change of advertised value.

```
js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp void registerValueCallback( YDisplayUpdateCallback callback)
pas procedure registerValueCallback( callback: TGenericUpdateCallback)
vb procedure registerValueCallback( ByVal callback As GenericUpdateCallback)
cs void registerValueCallback( UpdateCallback callback)
java void registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
m -(void) registerValueCallback : (YFunctionUpdateCallback) callback
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

#### Parameters :

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

**vsource→set\_logicalName()**  
**vsource→setLogicalName()** vsource→  
**set\_logicalName( )**

**YVSource**

Changes the logical name of the voltage source.

js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	YVSource <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 voltage source

**Returns :**

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

**vsouce→set(userData)****YVSource****vsouce→setUserData()vsouce→set(userData( )**

Stores a user context provided as argument in the userData attribute of the function.

```
js function set(userData( data)
php function set(userData( $data)
cpp void set(userData( void* data)
m -(void) setUserData : (void*) data
pas procedure set(userData( data: Tobject)
vb procedure set(userData( ByVal data As Object)
cs void set(userData( object data)
java void set(userData( Object data)
py def set(userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

**Parameters :**

**data** any kind of object to be stored

**vsource→set\_voltage()****YVSource****vsource→setVoltage()vsource→set\_voltage( )**

Tunes the device output voltage (milliVolts).

**js** function **set\_voltage( newval)****php** function **set\_voltage( \$newval)****cpp** int **set\_voltage( int newval)****m** -(int) setVoltage : (int) **newval****pas** function **set\_voltage( newval: LongInt): integer****vb** function **set\_voltage( ByVal newval As Integer) As Integer****cs** int **set\_voltage( int newval)****java** int **set\_voltage( int newval)****py** def **set\_voltage( newval)****cmd** YVSource **target set\_voltage newval****Parameters :****newval** an integer**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**vsource→voltageMove()****YVSource**

Performs a smooth move at constant speed toward a given value.

```
js function voltageMove( target, ms_duration)
php function voltageMove( $target, $ms_duration)
cpp int voltageMove( int target, int ms_duration)
m -(int) voltageMove : (int) target : (int) ms_duration
pas function voltageMove( target: integer, ms_duration: integer): integer
vb function voltageMove( ByVal target As Integer,
                           ByVal ms_duration As Integer) As Integer
cs int voltageMove( int target, int ms_duration)
java int voltageMove( int target, int ms_duration)
py def voltageMove( target, ms_duration)
cmd YVSource target voltageMove target ms_duration
```

**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;
require_once('yocto_wakeupmonitor.php');
php #include "yocto_wakeupmonitor.h"
cpp #import "yocto_wakeupmonitor.h"
m uses yocto_wakeupmonitor;
pas yocto_wakeupmonitor.vb
cs yocto_wakeupmonitor.cs
java import com.yoctopuce.YoctoAPI.YWakeUpMonitor;
py from yocto_wakeupmonitor import *

```

### Global functions

#### **yFindWakeUpMonitor(func)**

Retrieves a monitor for a given identifier.

#### **yFirstWakeUpMonitor()**

Starts the enumeration of monitors currently accessible.

### YWakeUpMonitor methods

#### **wakeupmonitor→describe()**

Returns a short text that describes unambiguously the instance of the monitor in the form TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **wakeupmonitor→get\_advertisedValue()**

Returns the current value of the monitor (no more than 6 characters).

#### **wakeupmonitor→get\_errorMessage()**

Returns the error message of the latest error with the monitor.

#### **wakeupmonitor→get\_errorType()**

Returns the numerical error code of the latest error with the monitor.

#### **wakeupmonitor→get\_friendlyName()**

Returns a global identifier of the monitor in the format MODULE \_ NAME . FUNCTION \_ NAME.

#### **wakeupmonitor→get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### **wakeupmonitor→get\_functionId()**

Returns the hardware identifier of the monitor, without reference to the module.

#### **wakeupmonitor→get\_hardwareId()**

Returns the unique hardware identifier of the monitor in the form SERIAL . FUNCTIONID.

#### **wakeupmonitor→get\_logicalName()**

Returns the logical name of the monitor.

#### **wakeupmonitor→get\_module()**

Gets the YModule object for the device on which the function is located.

#### **wakeupmonitor→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

#### **wakeupmonitor→get\_nextWakeUp()**

Returns the next scheduled wake up date/time (UNIX format)
<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()yFindWakeUpMonitor( )

## YWakeUpMonitor

Retrieves a monitor for a given identifier.

js	function <b>yFindWakeUpMonitor( func)</b>
node.js	function <b>FindWakeUpMonitor( func)</b>
php	function <b>yFindWakeUpMonitor( \$func)</b>
cpp	<b>YWakeUpMonitor*</b> <b>yFindWakeUpMonitor( const string&amp; func)</b>
m	<b>YWakeUpMonitor*</b> <b>yFindWakeUpMonitor( NSString* func)</b>
pas	function <b>yFindWakeUpMonitor( func: string): TYWakeUpMonitor</b>
vb	function <b>yFindWakeUpMonitor( ByVal func As String) As YWakeUpMonitor</b>
cs	<b>YWakeUpMonitor</b> <b>FindWakeUpMonitor( string func)</b>
java	<b>YWakeUpMonitor</b> <b>FindWakeUpMonitor( String func)</b>
py	def <b>FindWakeUpMonitor( 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()****YWakeUpMonitor****yFirstWakeUpMonitor()yFirstWakeUpMonitor( )**

Starts the enumeration of monitors currently accessible.

js	function <b>yFirstWakeUpMonitor( )</b>
node.js	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	- <b>(NSString*) 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	<b>def 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)=RELAYL01-123456.relay1 if the module is already connected or Relay(BadCustomeName.relay1)=unresolved if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

**Returns :**

a string that describes the monitor (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

wakeupmonitor→get\_advertisedValue()  
wakeupmonitor→advertisedValue(wakeupmonitor  
→get\_advertisedValue())

YWakeUpMonitor

Returns the current value of the monitor (no more than 6 characters).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YWakeUpMonitor target get_advertisedValue
```

**Returns :**

a string corresponding to the current value of the monitor (no more than 6 characters). On failure, throws an exception or returns Y\_ADVERTISEDVALUE\_INVALID.

**wakeupmonitor→getErrorMessage()****YWakeUpMonitor****wakeupmonitor→errorMessage()wakeupmonitor→  
getErrorMessage( )**

Returns the error message of the latest error with the monitor.

js	function getErrorMessage( )
node.js	function getErrorMessage( )
php	function getErrorMessage( )
cpp	string getErrorMessage( )
m	-(NSString*) errorMessage
pas	function getErrorMessage( ): string
vb	function getErrorMessage( ) As String
cs	string getErrorMessage( )
java	String getErrorMessage( )
py	def getErrorMessage( )

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.

**js** function **get\_errorType( )****nodejs** function **get\_errorType( )****php** function **get\_errorType( )****cpp** YRETCODE **get\_errorType( )****pas** function **get\_errorType( )**: YRETCODE**vb** function **get\_errorType( )** As YRETCODE**cs** YRETCODE **get\_errorType( )****java** int **get\_errorType( )****py** def **get\_errorType( )**

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a number corresponding to the code of the latest error that 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 get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the monitor if they are defined, otherwise the serial number of the module and the hardware identifier of the monitor (for exemple: MyCustomName.relay1)

**Returns :**

a string that uniquely identifies the monitor using logical names (ex: MyCustomName.relay1) On failure, throws an exception or returns Y\_FRIENDLYNAME\_INVALID.

**wakeupmonitor→get\_functionDescriptor()****YWakeUpMonitor****wakeupmonitor→functionDescriptor()****wakeupmonitor→get\_functionDescriptor( )**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor( )</b>
node.js	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	<b>def 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>
node.js	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( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def 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()****YWakeUpMonitor****wakeupmonitor→hardwareId()wakeupmonitor→  
get\_hardwareId( )**

Returns the unique hardware identifier of the monitor in the form SERIAL.FUNCTIONID.

js	function get_hardwareId( )
nodejs	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the monitor. (for example RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the monitor (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**wakeupmonitor→get\_logicalName()****YWakeUpMonitor****wakeupmonitor→logicalName()wakeupmonitor→  
get\_logicalName( )**

Returns the logical name of the monitor.

js	function get_logicalName( )
nodejs	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YWakeUpMonitor target get_logicalName

**Returns :**

a string corresponding to the logical name of the monitor. On failure, throws an exception or returns Y\_LOGICALNAME\_INVALID.

wakeupmonitor→get\_module()  
wakeupmonitor→module()wakeupmonitor→  
get\_module()

YWakeUpMonitor

Gets the `YModule` object for the device on which the function is located.

js	function get_module( )
nodejs	function get_module( )
php	function get_module( )
cpp	YModule * get_module( )
m	-(YModule*) module
pas	function get_module( ): TYModule
vb	function get_module( ) As YModule
cs	YModule get_module( )
java	YModule get_module( )
py	def get_module( )

If the function cannot be located on any module, the returned instance of `YModule` is not shown as online.

**Returns :**

an instance of `YModule`

**wakeupmonitor→get\_module\_async()****YWakeUpMonitor****wakeupmonitor→module\_async()**

Gets the YModule object for the device on which the function is located (asynchronous version).

```
js   function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

wakeupmonitor→get\_nextWakeUp()  
wakeupmonitor→nextWakeUp()wakeupmonitor→  
get\_nextWakeUp( )

YWakeUpMonitor

Returns the next scheduled wake up date/time (UNIX format)

```
js function get_nextWakeUp( )
nodejs function get_nextWakeUp( )
php function get_nextWakeUp( )
cpp s64 get_nextWakeUp( )
m -(s64) nextWakeUp
pas function get_nextWakeUp( ): int64
vb function get_nextWakeUp( ) As Long
cs long get_nextWakeUp( )
java long get_nextWakeUp( )
py def get_nextWakeUp( )
```

**Returns :**

an integer corresponding to the next scheduled wake up date/time (UNIX format)

On failure, throws an exception or returns Y\_NEXTWAKEUP\_INVALID.

**wakeupmonitor→get\_powerDuration()****YWakeUpMonitor****wakeupmonitor→powerDuration()wakeupmonitor→  
get\_powerDuration()**

Returns the maximal wake up time (in seconds) before automatically going to sleep.

js	function <b>get_powerDuration( )</b>
node.js	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 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 get_sleepCountdown( )
nodejs function get_sleepCountdown( )
php function get_sleepCountdown( )
cpp int get_sleepCountdown( )
m -(int) sleepCountdown
pas function get_sleepCountdown( ): LongInt
vb function get_sleepCountdown( ) As Integer
cs int get_sleepCountdown( )
java int get_sleepCountdown( )
py def get_sleepCountdown( )
cmd YWakeUpMonitor target get_sleepCountdown
```

**Returns :**

an integer corresponding to the delay before the next sleep period

On failure, throws an exception or returns Y\_SLEEPCOUNTDOWN\_INVALID.

**wakeupmonitor→get(userData())****YWakeUpMonitor****wakeupmonitor→userData()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*) <b>get(userData)</b> {
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()  
wakeupmonitor→wakeUpReason()wakeupmonitor→  
get\_wakeUpReason()

YWakeUpMonitor

Returns the latest wake up reason.

js	function get_wakeUpReason( )
nodejs	function get_wakeUpReason( )
php	function get_wakeUpReason( )
cpp	Y_WAKEUPREASON_enum get_wakeUpReason( )
m	-(Y_WAKEUPREASON_enum) wakeUpReason
pas	function get_wakeUpReason( ): Integer
vb	function get_wakeUpReason( ) As Integer
cs	int get_wakeUpReason( )
java	int get_wakeUpReason( )
py	def get_wakeUpReason( )
cmd	YWakeUpMonitor target get_wakeUpReason

#### Returns :

a value among Y\_WAKEUPREASON\_USBPOWER, Y\_WAKEUPREASON\_EXTPOWER,  
Y\_WAKEUPREASON\_ENDOFSLEEP, Y\_WAKEUPREASON\_EXTSIG1,  
Y\_WAKEUPREASON\_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

js	function get_wakeUpState( )
node.js	function get_wakeUpState( )
php	function get_wakeUpState( )
cpp	Y_WAKEUPSTATE_enum get_wakeUpState( )
m	-(Y_WAKEUPSTATE_enum) wakeUpState
pas	function get_wakeUpState( ): Integer
vb	function get_wakeUpState( ) As Integer
cs	int get_wakeUpState( )
java	int get_wakeUpState( )
py	def get_wakeUpState( )

**Returns :**

either Y\_WAKEUPSTATE\_SLEEPING or Y\_WAKEUPSTATE\_AWAKE, according to the current state of the monitor

On failure, throws an exception or returns Y\_WAKEUPSTATE\_INVALID.

**wakeupmonitor→isOnline()wakeupmonitor→  
isOnline()****YWakeUpMonitor**

Checks if the monitor is currently reachable, without raising any error.

js	function <b>isOnline()</b>
node.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 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)
node.js	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( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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()****YWakeUpMonitor**

Continues the enumeration of monitors started using `yFirstWakeUpMonitor()`.

js	function <b>nextWakeUpMonitor()</b>
node.js	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.

<code>js</code>	<code>function registerValueCallback( callback)</code>
<code>node.js</code>	<code>function registerValueCallback( callback)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>int registerValueCallback( YWakeUpMonitorValueCallback callback)</code>
<code>m</code>	<code>-(int) registerValueCallback : (YWakeUpMonitorValueCallback) callback</code>
<code>pas</code>	<code>function registerValueCallback( callback: TYWakeUpMonitorValueCallback): LongInt</code>
<code>vb</code>	<code>function registerValueCallback( ) As Integer</code>
<code>cs</code>	<code>int registerValueCallback( ValueCallback callback)</code>
<code>java</code>	<code>int registerValueCallback( UpdateCallback callback)</code>
<code>py</code>	<code>def registerValueCallback( callback)</code>

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

### Parameters :

**callback** the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

**wakeupmonitor→resetSleepCountDown()****YWakeUpMonitor****wakeupmonitor→resetSleepCountDown( )**

Resets the sleep countdown.

js	function <b>resetSleepCountDown()</b>
node.js	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	<b>YWakeUpMonitor target resetSleepCountDown</b>

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**wakeupmonitor→set\_logicalName()** YWakeUpMonitor  
**wakeupmonitor→setLogicalName()** *wakeupmonitor*  
*→set\_logicalName( )*

---

Changes the logical name of the monitor.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YWakeUpMonitor 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 monitor.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

wakeupmonitor→**set\_nextWakeUp()**  
wakeupmonitor→**setNextWakeUp()**wakeupmonitor  
→**set\_nextWakeUp( )**

YWakeUpMonitor

Changes the days of the week when a wake up must take place.

js	function <b>set_nextWakeUp( newval)</b>
nodejs	function <b>set_nextWakeUp( newval)</b>
php	function <b>set_nextWakeUp( \$newval)</b>
cpp	int <b>set_nextWakeUp( s64 newval)</b>
m	-(int) <b>setNextWakeUp : (s64) newval</b>
pas	function <b>set_nextWakeUp( newval: int64): integer</b>
vb	function <b>set_nextWakeUp( ByVal newval As Long) As Integer</b>
cs	int <b>set_nextWakeUp( long newval)</b>
java	int <b>set_nextWakeUp( long newval)</b>
py	def <b>set_nextWakeUp( newval)</b>
cmd	YWakeUpMonitor <b>target set_nextWakeUp 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()**  
wakeupmonitor→**setPowerDuration()**  
wakeupmonitor→**set\_powerDuration( )**

**YWakeUpMonitor**

Changes the maximal wake up time (seconds) before automatically going to sleep.

js	function <b>set_powerDuration( newval)</b>
node.js	function <b>set_powerDuration( newval)</b>
php	function <b>set_powerDuration( \$newval)</b>
cpp	int <b>set_powerDuration( int newval)</b>
m	-(int) <b>setPowerDuration : (int) newval</b>
pas	function <b>set_powerDuration( newval: LongInt): integer</b>
vb	function <b>set_powerDuration( ByVal newval As Integer) As Integer</b>
cs	int <b>set_powerDuration( int newval)</b>
java	int <b>set_powerDuration( int newval)</b>
py	def <b>set_powerDuration( newval)</b>
cmd	<b>YWakeUpMonitor target set_powerDuration 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 set_sleepCountdown( newval)
nodejs function set_sleepCountdown( newval)
php function set_sleepCountdown( $newval)
cpp int set_sleepCountdown( int newval)
m -(int) setSleepCountdown : (int) newval
pas function set_sleepCountdown( newval: LongInt): integer
vb function set_sleepCountdown( ByVal newval As Integer) As Integer
cs int set_sleepCountdown( int newval)
java int set_sleepCountdown( int newval)
py def set_sleepCountdown( newval)
cmd YWakeUpMonitor target set_sleepCountdown newval
```

**Parameters :**

**newval** an integer corresponding to the delay before the next sleep period

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**wakeupmonitor→set(userData)****YWakeUpMonitor****wakeupmonitor→setUserData()wakeupmonitor→  
set(userData)**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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()****YWakeUpMonitor**

Goes to sleep until the next wake up condition is met, the RTC time must have been set before calling this function.

```
js   function sleep( secBeforeSleep)
nodejs function sleep( secBeforeSleep)
php  function sleep( $secBeforeSleep)
cpp   int sleep( int secBeforeSleep)
m    -(int) sleep : (int) secBeforeSleep
pas   function sleep( secBeforeSleep: LongInt): LongInt
vb    function sleep( ) As Integer
cs   int sleep( int secBeforeSleep)
java  int sleep( int secBeforeSleep)
py    def sleep( secBeforeSleep)
cmd   YWakeUpMonitor target sleep secBeforeSleep
```

**Parameters :**

**secBeforeSleep** number of seconds before going into sleep mode,

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

wakeupmonitor→**sleepFor()**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.

<code>js</code>	function <b>sleepFor( secUntilWakeUp, secBeforeSleep)</b>
<code>node.js</code>	function <b>sleepFor( secUntilWakeUp, secBeforeSleep)</b>
<code>php</code>	function <b>sleepFor( \$secUntilWakeUp, \$secBeforeSleep)</b>
<code>cpp</code>	int <b>sleepFor( int secUntilWakeUp, int secBeforeSleep)</b>
<code>m</code>	- <b>(int) sleepFor : (int) secUntilWakeUp : (int) secBeforeSleep</b>
<code>pas</code>	function <b>sleepFor( secUntilWakeUp: LongInt,</b> <b>secBeforeSleep: LongInt): LongInt</b>
<code>vb</code>	function <b>sleepFor( ) As Integer</b>
<code>cs</code>	int <b>sleepFor( int secUntilWakeUp, int secBeforeSleep)</b>
<code>java</code>	int <b>sleepFor( int secUntilWakeUp, int secBeforeSleep)</b>
<code>py</code>	<b>def sleepFor( secUntilWakeUp, secBeforeSleep)</b>
<code>cmd</code>	<b>YWakeUpMonitor target sleepFor secUntilWakeUp 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.

<b>js</b>	function <b>sleepUntil(</b> wakeUpTime, secBeforeSleep)
<b>nodejs</b>	function <b>sleepUntil(</b> wakeUpTime, secBeforeSleep)
<b>php</b>	function <b>sleepUntil(</b> \$wakeUpTime, \$secBeforeSleep)
<b>cpp</b>	int <b>sleepUntil(</b> int wakeUpTime, int secBeforeSleep)
<b>m</b>	- <b>(int) sleepUntil : (int) wakeUpTime : (int) secBeforeSleep</b>
<b>pas</b>	function <b>sleepUntil(</b> wakeUpTime: LongInt, secBeforeSleep: LongInt): LongInt
<b>vb</b>	function <b>sleepUntil( ) As Integer</b>
<b>cs</b>	int <b>sleepUntil(</b> int wakeUpTime, int secBeforeSleep)
<b>java</b>	int <b>sleepUntil(</b> int wakeUpTime, int secBeforeSleep)
<b>py</b>	def <b>sleepUntil(</b> wakeUpTime, secBeforeSleep)
<b>cmd</b>	<b>YWakeUpMonitor target sleepUntil wakeUpTime 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.

wakeupmonitor → **wakeUp()** wakeupmonitor →  
**wakeUp( )**

**YWakeUpMonitor**

Forces a wake up.

```
js function wakeUp( )
node.js function wakeUp( )
php function wakeUp( )
cpp int wakeUp( )
m -(int) wakeUp
pas function wakeUp( ): LongInt
vb function wakeUp( ) As Integer
cs int wakeUp( )
java int wakeUp( )
py def wakeUp( )
cmd YWakeUpMonitor target wakeUp
```

## 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');
cpp	#include "yocto_wakeupschedule.h"
m	#import "yocto_wakeupschedule.h"
pas	uses yocto_wakeupschedule;
vb	yocto_wakeupschedule.vb
cs	yocto_wakeupschedule.cs
java	import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py	from yocto_wakeupschedule import *

### Global functions

#### yFindWakeUpSchedule(func)

Retrieves a wake up schedule for a given identifier.

#### yFirstWakeUpSchedule()

Starts the enumeration of wake up schedules currently accessible.

### YWakeUpSchedule methods

#### wakeupschedule→describe()

Returns a short text that describes unambiguously the instance of the wake up schedule in the form  
TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### wakeupschedule→get\_advertisedValue()

Returns the current value of the wake up schedule (no more than 6 characters).

#### wakeupschedule→get\_errorMessage()

Returns the error message of the latest error with the wake up schedule.

#### wakeupschedule→get\_errorType()

Returns the numerical error code of the latest error with the wake up schedule.

#### wakeupschedule→get\_friendlyName()

Returns a global identifier of the wake up schedule in the format MODULE\_NAME . FUNCTION\_NAME.

#### wakeupschedule→get\_functionDescriptor()

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### wakeupschedule→get\_functionId()

Returns the hardware identifier of the wake up schedule, without reference to the module.

#### wakeupschedule→get\_hardwareId()

Returns the unique hardware identifier of the wake up schedule in the form SERIAL . FUNCTIONID.

#### wakeupschedule→get\_hours()

Returns the hours scheduled for wake up.

#### wakeupschedule→get\_logicalName()

Returns the logical name of the wake up schedule.

#### wakeupschedule→get\_minutes()

Returns all the minutes of each hour that are scheduled for wake up.

#### wakeupschedule→get\_minutesA()

Returns the minutes in the 00-29 interval of each hour scheduled for wake up.

**wakeupschedule→get\_minutesB()**

Returns the minutes in the 30-59 interval of each hour scheduled for wake up.

**wakeupschedule→get\_module()**

Gets the YModule object for the device on which the function is located.

**wakeupschedule→get\_module\_async(callback, context)**

Gets the YModule object for the device on which the function is located (asynchronous version).

**wakeupschedule→get\_monthDays()**

Returns the days of the month scheduled for wake up.

**wakeupschedule→get\_months()**

Returns the months scheduled for wake up.

**wakeupschedule→get\_nextOccurrence()**

Returns the date/time (seconds) of the next wake up occurrence

**wakeupschedule→get\_userData()**

Returns the value of the userData attribute, as previously stored using method set(userData).

**wakeupschedule→get\_weekDays()**

Returns the days of the week scheduled for wake up.

**wakeupschedule→isOnline()**

Checks if the wake up schedule is currently reachable, without raising any error.

**wakeupschedule→isOnline\_async(callback, context)**

Checks if the wake up schedule is currently reachable, without raising any error (asynchronous version).

**wakeupschedule→load(msValidity)**

Preloads the wake up schedule cache with a specified validity duration.

**wakeupschedule→load\_async(msValidity, callback, context)**

Preloads the wake up schedule cache with a specified validity duration (asynchronous version).

**wakeupschedule→nextWakeUpSchedule()**

Continues the enumeration of wake up schedules started using yFirstWakeUpSchedule( ).

**wakeupschedule→registerValueCallback(callback)**

Registers the callback function that is invoked on every change of advertised value.

**wakeupschedule→set\_hours(newval)**

Changes the hours when a wake up must take place.

**wakeupschedule→set\_logicalName(newval)**

Changes the logical name of the wake up schedule.

**wakeupschedule→set\_minutes(bitmap)**

Changes all the minutes where a wake up must take place.

**wakeupschedule→set\_minutesA(newval)**

Changes the minutes in the 00-29 interval when a wake up must take place.

**wakeupschedule→set\_minutesB(newval)**

Changes the minutes in the 30-59 interval when a wake up must take place.

**wakeupschedule→set\_monthDays(newval)**

Changes the days of the month when a wake up must take place.

**wakeupschedule→set\_months(newval)**

Changes the months when a wake up must take place.

**wakeupschedule→set\_userData(data)**

Stores a user context provided as argument in the userData attribute of the function.

**wakeupschedule→set\_weekDays(newval)**

Changes the days of the week when a wake up must take place.

**wakeupschedule→wait\_async(callback, context)**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

## YWakeUpSchedule.FindWakeUpSchedule() yFindWakeUpSchedule()yFindWakeUpSchedule()

**YWakeUpSchedule**

Retrieves a wake up schedule for a given identifier.

js	function <b>yFindWakeUpSchedule( func)</b>
node.js	function <b>FindWakeUpSchedule( func)</b>
php	function <b>yFindWakeUpSchedule( \$func)</b>
cpp	YWakeUpSchedule* <b>yFindWakeUpSchedule( const string&amp; func)</b>
m	YWakeUpSchedule* <b>yFindWakeUpSchedule( NSString* func)</b>
pas	function <b>yFindWakeUpSchedule( func: string): TYWakeUpSchedule</b>
vb	function <b>yFindWakeUpSchedule( ByVal func As String) As YWakeUpSchedule</b>
cs	YWakeUpSchedule <b>FindWakeUpSchedule( string func)</b>
java	YWakeUpSchedule <b>FindWakeUpSchedule( String func)</b>
py	def <b>FindWakeUpSchedule( 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()yFirstWakeUpSchedule()****YWakeUpSchedule**

Starts the enumeration of wake up schedules currently accessible.

js	function <b>yFirstWakeUpSchedule( )</b>
node.js	function <b>FirstWakeUpSchedule( )</b>
php	function <b>yFirstWakeUpSchedule( )</b>
cpp	YWakeUpSchedule* <b>yFirstWakeUpSchedule( )</b>
m	YWakeUpSchedule* <b>yFirstWakeUpSchedule( )</b>
pas	function <b>yFirstWakeUpSchedule( ): TYWakeUpSchedule</b>
vb	function <b>yFirstWakeUpSchedule( ) As YWakeUpSchedule</b>
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	- <b>(NSString*) describe</b>
pas	function <b>describe( )</b> : string
vb	function <b>describe( ) As String</b>
cs	string <b>describe( )</b>
java	String <b>describe( )</b>
py	<b>def 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)=RELAYL01-123456.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)=RELAYL01-123456.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>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
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	<b>def get_advertisedValue( )</b>
cmd	YWakeUpSchedule <b>target 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()  
wakeupschedule→errorMessage()wakeupschedule  
→get\_errorMessage( )

YWakeUpSchedule

Returns the error message of the latest error with the wake up schedule.

js	function get_errorMessage( )
nodejs	function get_errorMessage( )
php	function get_errorMessage( )
cpp	string get_errorMessage( )
m	-(NSString*) errorMessage
pas	function get_errorMessage( ): string
vb	function get_errorMessage( ) As String
cs	string get_errorMessage( )
java	String get_errorMessage( )
py	def get_errorMessage( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that 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 get_errorType( )
node.js	function get_errorType( )
php	function get_errorType( )
cpp	YRETCODE get_errorType( )
pas	function get_errorType( ): YRETCODE
vb	function get_errorType( ) As YRETCODE
cs	YRETCODE get_errorType( )
java	int get_errorType( )
py	def get_errorType( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a number corresponding to the code of the latest error that 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 **get\_friendlyName( )****nodejs** function **get\_friendlyName( )****php** function **get\_friendlyName( )****cpp** string **get\_friendlyName( )****m** -(NSString\*) friendlyName**cs** string **get\_friendlyName( )****java** String **get\_friendlyName( )****py** def **get\_friendlyName( )**

The returned string uses the logical names of the module and of the wake up schedule if they are defined, otherwise the serial number of the module and the hardware identifier of the wake up schedule (for exemple: MyCustomName.relay1)

**Returns :**

a string that uniquely identifies the wake up schedule using logical names (ex: MyCustomName.relay1) On failure, throws an exception or returns Y\_FRIENDLYNAME\_INVALID.

---

wakeupschedule→get_functionDescriptor()	YWakeUpSchedule
wakeupschedule→functionDescriptor()	
wakeupschedule→get_functionDescriptor( )	

---

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

```
js    function get_functionDescriptor( )  
node.js function get_functionDescriptor( )  
php   function get_functionDescriptor( )  
cpp   YFUN_DESCR get_functionDescriptor( )  
m     -(YFUN_DESCR) functionDescriptor  
pas   function get_functionDescriptor( ): YFUN_DESCR  
vb    function get_functionDescriptor( ) As YFUN_DESCR  
cs    YFUN_DESCR get_functionDescriptor( )  
java  String get_functionDescriptor( )  
py    def get_functionDescriptor( )
```

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

**Returns :**

an identifier of type YFUN\_DESCR. If the function has never been contacted, the returned value is Y\_FUNCTIONDESCRIPTOR\_INVALID.

**wakeupschedule→get\_functionId()****YWakeUpSchedule****wakeupschedule→functionId()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>
node.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( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	<b>def 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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the wake up schedule (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

wakeupschedule→get\_hours()  
wakeupschedule→hours()wakeupschedule→  
get\_hours( )

YWakeUpSchedule

Returns the hours scheduled for wake up.

js	function get_hours( )
nodejs	function get_hours( )
php	function get_hours( )
cpp	int get_hours( )
m	-(int) hours
pas	function get_hours( ): LongInt
vb	function get_hours( ) As Integer
cs	int get_hours( )
java	int get_hours( )
py	def get_hours( )
cmd	YWakeUpSchedule target get_hours

**Returns :**

an integer corresponding to the hours scheduled for wake up

On failure, throws an exception or returns Y\_HOURS\_INVALID.

**wakeupschedule→get\_logicalName()****YWakeUpSchedule****wakeupschedule→logicalName()wakeupschedule→  
get\_logicalName( )**

Returns the logical name of the wake up schedule.

js	function <b>get_logicalName( )</b>
node.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	<b>def get_logicalName( )</b>
cmd	YWakeUpSchedule <b>target 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 **get\_minutes( )****nodejs** function **get\_minutes( )****php** function **get\_minutes( )****cpp** s64 **get\_minutes( )****m** -(s64) minutes**pas** function **get\_minutes( ): int64****vb** function **get\_minutes( ) As Long****cs** long **get\_minutes( )****java** long **get\_minutes( )****py** def **get\_minutes( )****cmd** YWakeUpSchedule **target get\_minutes**

**wakeupschedule→get\_minutesA()****YWakeUpSchedule****wakeupschedule→minutesA()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	<b>YWakeUpSchedule target 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 get_minutesB( )`**nodejs**`function get_minutesB( )`**php**`function get_minutesB( )`**cpp**`int get_minutesB( )`**m**`-(int) minutesB`**pas**`function get_minutesB( ): LongInt`**vb**`function get_minutesB( ) As Integer`**cs**`int get_minutesB( )`**java**`int get_minutesB( )`**py**`def get_minutesB( )`**cmd**`YWakeUpSchedule target get_minutesB`**Returns :**

an integer corresponding to the minutes in the 30-59 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> : 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

**wakeupschedule→get\_module\_async()****YWakeUpSchedule****wakeupschedule→module\_async()**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

```
js  function get_module_async( callback, context)
node.js function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

wakeupschedule→get\_monthDays()

YWakeUpSchedule

wakeupschedule→monthDays()wakeupschedule→  
get\_monthDays( )

Returns the days of the month scheduled for wake up.

js	function get_monthDays( )
nodejs	function get_monthDays( )
php	function get_monthDays( )
cpp	int get_monthDays( )
m	-(int) monthDays
pas	function get_monthDays( ): LongInt
vb	function get_monthDays( ) As Integer
cs	int get_monthDays( )
java	int get_monthDays( )
py	def get_monthDays( )
cmd	YWakeUpSchedule target get_monthDays

**Returns :**

an integer corresponding to the days of the month scheduled for wake up

On failure, throws an exception or returns Y\_MONTHDAYS\_INVALID.

wakeupschedule→get\_months()  
wakeupschedule→months()wakeupschedule→  
get\_months( )

YWakeUpSchedule

Returns the months scheduled for wake up.

```
js function get_months( )
nodejs function get_months( )
php function get_months( )
cpp int get_months( )
m -(int) months
pas function get_months( ): LongInt
vb function get_months( ) As Integer
cs int get_months( )
java int get_months( )
py def get_months( )
cmd YWakeUpSchedule target get_months
```

**Returns :**

an integer corresponding to the months scheduled for wake up

On failure, throws an exception or returns Y\_MONTHS\_INVALID.

wakeupschedule→get\_nextOccurence()  
wakeupschedule→nextOccurence()wakeupschedule  
→get\_nextOccurence( )

YWakeUpSchedule

Returns the date/time (seconds) of the next wake up occurence

js	function get_nextOccurence( )
node.js	function get_nextOccurence( )
php	function get_nextOccurence( )
cpp	s64 get_nextOccurence( )
m	-(s64) nextOccurence
pas	function get_nextOccurence( ): int64
vb	function get_nextOccurence( ) As Long
cs	long get_nextOccurence( )
java	long get_nextOccurence( )
py	def get_nextOccurence( )

**Returns :**

an integer corresponding to the date/time (seconds) of the next wake up occurence

On failure, throws an exception or returns Y\_NEXTOCCURENCE\_INVALID.

**wakeupschedule→get(userData)****YWakeUpSchedule****wakeupschedule→userData()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) <b>weekDays</b>
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	<b>YWakeUpSchedule target 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>
node.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 wake up schedule in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the wake up schedule.

**Returns :**

true if the wake up schedule can be reached, and false otherwise

**wakeupschedule→isOnline\_async()****YWakeUpSchedule**

Checks if the wake up schedule is currently reachable, without raising any error (asynchronous version).

```
js function isOnline_async( callback, context)
nodejs function isOnline_async( callback, context)
```

If there is a cached value for the wake up schedule in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**wakeupschedule→load()****YWakeUpSchedule**

Preloads the wake up schedule cache with a specified validity duration.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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()****YWakeUpSchedule****wakeupschedule→nextWakeUpSchedule( )**

Continues the enumeration of wake up schedules started using `yFirstWakeUpSchedule()`.

<code>js</code>	<code>function nextWakeUpSchedule( )</code>
<code>node.js</code>	<code>function nextWakeUpSchedule( )</code>
<code>php</code>	<code>function nextWakeUpSchedule( )</code>
<code>cpp</code>	<code>YWakeUpSchedule * nextWakeUpSchedule( )</code>
<code>m</code>	<code>-(YWakeUpSchedule*) nextWakeUpSchedule</code>
<code>pas</code>	<code>function nextWakeUpSchedule( ): TYWakeUpSchedule</code>
<code>vb</code>	<code>function nextWakeUpSchedule( ) As YWakeUpSchedule</code>
<code>cs</code>	<code>YWakeUpSchedule nextWakeUpSchedule( )</code>
<code>java</code>	<code>YWakeUpSchedule nextWakeUpSchedule( )</code>
<code>py</code>	<code>def nextWakeUpSchedule( )</code>

**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.

<code>js</code>	<code>function registerValueCallback( callback)</code>
<code>node.js</code>	<code>function registerValueCallback( callback)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>int registerValueCallback( YWakeUpScheduleValueCallback callback)</code>
<code>m</code>	<code>-(int) registerValueCallback : (YWakeUpScheduleValueCallback) callback</code>
<code>pas</code>	<code>function registerValueCallback( callback: TYWakeUpScheduleValueCallback): LongInt</code>
<code>vb</code>	<code>function registerValueCallback( ) As Integer</code>
<code>cs</code>	<code>int registerValueCallback( ValueCallback callback)</code>
<code>java</code>	<code>int registerValueCallback( UpdateCallback callback)</code>
<code>py</code>	<code>def registerValueCallback( callback)</code>

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()  
wakeupschedule→setHours()wakeupschedule→  
set\_hours( )

YWakeUpSchedule

Changes the hours when a wake up must take place.

```
js function set_hours( newval)
nodejs function set_hours( newval)
php function set_hours( $newval)
cpp int set_hours( int newval)
m -(int) setHours : (int) newval
pas function set_hours( newval: LongInt): integer
vb function set_hours( ByVal newval As Integer) As Integer
cs int set_hours( int newval)
java int set_hours( int newval)
py def set_hours( newval)
cmd YWakeUpSchedule target set_hours newval
```

**Parameters :**

**newval** an integer corresponding to the hours when a wake up must take place

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**wakeupschedule→set\_logicalName()**  
**wakeupschedule→setLogicalName()**  
**wakeupschedule→set\_logicalName( )**

**YWakeUpSchedule**

Changes the logical name of the wake up schedule.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YWakeUpSchedule 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 wake up schedule.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

wakeupschedule→set\_minutes()  
wakeupschedule→setMinutes()wakeupschedule→  
set\_minutes( )

YWakeUpSchedule

Changes all the minutes where a wake up must take place.

```
js function set_minutes( bitmap)
nodejs function set_minutes( bitmap)
php function set_minutes( $bitmap)
cpp int set_minutes( s64 bitmap)
m -(int) setMinutes : (s64) bitmap
pas function set_minutes( bitmap: int64): LongInt
vb function set_minutes( ) As Integer
cs int set_minutes( long bitmap)
java int set_minutes( long bitmap)
py def set_minutes( bitmap)
cmd YWakeUpSchedule target set_minutes bitmap
```

**Parameters :**

**bitmap** Minutes 00-59 of each hour scheduled for wake up.

**Returns :**

YAPI\_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

**wakeupschedule→set\_minutesA()****YWakeUpSchedule****wakeupschedule→setMinutesA()wakeupschedule→  
set\_minutesA( )**

Changes the minutes in the 00-29 interval when a wake up must take place.

<b>js</b>	function <b>set_minutesA( newval)</b>
<b>nodejs</b>	function <b>set_minutesA( newval)</b>
<b>php</b>	function <b>set_minutesA( \$newval)</b>
<b>cpp</b>	int <b>set_minutesA( int newval)</b>
<b>m</b>	-(int) setMinutesA : (int) newval
<b>pas</b>	function <b>set_minutesA( newval: LongInt): integer</b>
<b>vb</b>	function <b>set_minutesA( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_minutesA( int newval)</b>
<b>java</b>	int <b>set_minutesA( int newval)</b>
<b>py</b>	def <b>set_minutesA( newval)</b>
<b>cmd</b>	<b>YWakeUpSchedule target set_minutesA 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()  
wakeupschedule→setMinutesB()wakeupschedule→  
set\_minutesB()

YWakeUpSchedule

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( newval)</b>
node.js	function <b>set_monthDays( newval)</b>
php	function <b>set_monthDays( \$newval)</b>
cpp	int <b>set_monthDays( int newval)</b>
m	-(int) setMonthDays : (int) <b>newval</b>
pas	function <b>set_monthDays( newval: LongInt): integer</b>
vb	function <b>set_monthDays( ByVal newval As Integer) As Integer</b>
cs	int <b>set_monthDays( int newval)</b>
java	int <b>set_monthDays( int newval)</b>
py	def <b>set_monthDays( newval)</b>
cmd	<b>YWakeUpSchedule target set_monthDays 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()  
wakeupschedule→setMonths()wakeupschedule→  
set\_months( )

YWakeUpSchedule

Changes the months when a wake up must take place.

```
js function set_months( newval)
nodejs function set_months( newval)
php function set_months( $newval)
cpp int set_months( int newval)
m -(int) setMonths : (int) newval
pas function set_months( newval: LongInt): integer
vb function set_months( ByVal newval As Integer) As Integer
cs int set_months( int newval)
java int set_months( int newval)
py def set_months( newval)
cmd YWakeUpSchedule target set_months newval
```

**Parameters :**

**newval** an integer corresponding to the months when a wake up must take place

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**wakeupschedule→set(userData)****YWakeUpSchedule****wakeupschedule→setUserData()wakeupschedule→  
set(userData)**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData data: Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData 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()  
wakeupschedule→setWeekDays()wakeupschedule  
→set\_weekDays( )

YWakeUpSchedule

Changes the days of the week when a wake up must take place.

js	function set_weekDays( newval)
nodejs	function set_weekDays( newval)
php	function set_weekDays( \$newval)
cpp	int set_weekDays( int newval)
m	-(int) setWeekDays : (int) newval
pas	function set_weekDays( newval: LongInt): integer
vb	function set_weekDays( ByVal newval As Integer) As Integer
cs	int set_weekDays( int newval)
java	int set_weekDays( int newval)
py	def set_weekDays( newval)
cmd	YWakeUpSchedule target set_weekDays newval

**Parameters :**

**newval** an integer corresponding to the days of the week when a wake up must take place

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**wakeupschedule→wait\_async()****YWakeUpSchedule**

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

```
js  function wait_async( callback, context)
nodejs function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

**Parameters :**

**callback** callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing.

## 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');
cpp #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.

### 3. Reference

#### **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()yFindWatchdog( )****YWatchdog**

Retrieves a watchdog for a given identifier.

<code>js</code>	<code>function yFindWatchdog( func)</code>
<code>node.js</code>	<code>function FindWatchdog( func)</code>
<code>php</code>	<code>function yFindWatchdog( \$func)</code>
<code>cpp</code>	<code>YWatchdog* yFindWatchdog( const string&amp; func)</code>
<code>m</code>	<code>YWatchdog* yFindWatchdog( NSString* func)</code>
<code>pas</code>	<code>function yFindWatchdog( func: string): TYWatchdog</code>
<code>vb</code>	<code>function yFindWatchdog( ByVal func As String) As YWatchdog</code>
<code>cs</code>	<code>YWatchdog FindWatchdog( string func)</code>
<code>java</code>	<code>YWatchdog FindWatchdog( String func)</code>
<code>py</code>	<code>def FindWatchdog( 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 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()yFirstWatchdog( )**

Starts the enumeration of watchdog currently accessible.

js	function <b>yFirstWatchdog( )</b>
node.js	function <b>FirstWatchdog( )</b>
php	function <b>yFirstWatchdog( )</b>
cpp	YWatchdog* <b>yFirstWatchdog( )</b>
m	YWatchdog* <b>yFirstWatchdog( )</b>
pas	function <b>yFirstWatchdog( ): TYWatchdog</b>
vb	function <b>yFirstWatchdog( ) As YWatchdog</b>
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() watchdog→ delayedPulse()

YWatchdog

Schedules a pulse.

```

js   function delayedPulse( ms_delay, ms_duration)
nodejs function delayedPulse( ms_delay, ms_duration)
php  function delayedPulse( $ms_delay, $ms_duration)
cpp   int delayedPulse( int ms_delay, int ms_duration)
m    -(int) delayedPulse : (int) ms_delay : (int) ms_duration
pas   function delayedPulse( ms_delay: LongInt, ms_duration: LongInt): integer
vb    function delayedPulse( ByVal ms_delay As Integer,
                           ByVal ms_duration As Integer) As Integer
cs    int delayedPulse( int ms_delay, int ms_duration)
java  int delayedPulse( int ms_delay, int ms_duration)
py    def delayedPulse( ms_delay, ms_duration)
cmd   YWatchdog target delayedPulse ms_delay ms_duration

```

### Parameters :

**ms\_delay** waiting time before the pulse, in millisecondes  
**ms\_duration** pulse duration, in millisecondes

### Returns :

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

**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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**watchdog→get\_advertisedValue()**  
**watchdog→advertisedValue()**  
**watchdog→get\_advertisedValue( )**

**YWatchdog**

Returns the current value of the watchdog (no more than 6 characters).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>node.js</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YWatchdog target 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()**  
**watchdog→autoStart()watchdog→**  
**get\_autoStart( )****YWatchdog**

Returns the watchdog runing state at module power up.

js	function get_autoStart( )
nodejs	function get_autoStart( )
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	<b>YWatchdog target 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()**  
**watchdog→countdown()****watchdog→get\_countdown( )****YWatchdog**

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) <b>countdown</b>
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 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()**  
**watchdog→errorMessage()** watchdog→  
**get\_errorMessage( )**

**YWatchdog**

Returns the error message of the latest error with the watchdog.

js	function get_errorMessage( )
nodejs	function get_errorMessage( )
php	function get_errorMessage( )
cpp	string get_errorMessage( )
m	-(NSString*) errorMessage
pas	function get_errorMessage( ): string
vb	function get_errorMessage( ) As String
cs	string get_errorMessage( )
java	String get_errorMessage( )
py	def get_errorMessage( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that occurred while using the watchdog object

**watchdog→get\_errorType()**  
**watchdog→errorType()watchdog→**  
**get\_errorType( )****YWatchdog**

---

Returns the numerical error code of the latest error with the watchdog.

js	function get_errorType( )
node.js	function get_errorType( )
php	function get_errorType( )
cpp	YRETCODE get_errorType( )
pas	function get_errorType( ): YRETCODE
vb	function get_errorType( ) As YRETCODE
cs	YRETCODE get_errorType( )
java	int get_errorType( )
py	def get_errorType( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a number corresponding to the code of the latest error that occurred while using the watchdog object

**watchdog→get\_friendlyName()**  
**watchdog→friendlyName()** watchdog→  
**get\_friendlyName( )**

**YWatchdog**

Returns a global identifier of the watchdog in the format MODULE\_NAME . FUNCTION\_NAME.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

The returned string uses the logical names of the module and of the 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()**  
**watchdog→functionDescriptor()watchdog→**  
**get\_functionDescriptor( )**

**YWatchdog**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor( )</b>
node.js	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.

**watchdog→get\_functionId()**  
**watchdog→functionId()watchdog→**  
**get\_functionId( )**

**YWatchdog**

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()**  
**watchdog→hardwareId()** watchdog→  
**get\_hardwareId( )**

**YWatchdog**

Returns the unique hardware identifier of the watchdog in the form SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.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	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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the watchdog (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

**watchdog→get\_logicalName()**  
**watchdog→logicalName()** watchdog→  
**get\_logicalName( )**

**YWatchdog**

Returns the logical name of the watchdog.

```
js function get_logicalName( )
nodejs function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YWatchdog target get_logicalName
```

**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>
node.js	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( ): int64</b>
vb	function <b>get_maxTimeOnStateA( ) As Long</b>
cs	long <b>get_maxTimeOnStateA( )</b>
java	long <b>get_maxTimeOnStateA( )</b>
py	def <b>get_maxTimeOnStateA( )</b>
cmd	<b>YWatchdog target get_maxTimeOnStateA</b>

Zero means no maximum time.

**Returns :**

an integer

On failure, throws an exception or returns Y\_MAXTIMEONSTATEA\_INVALID.

**watchdog→get\_maxTimeOnStateB()**  
**watchdog→maxTimeOnStateB()** watchdog→  
**get\_maxTimeOnStateB( )**

**YWatchdog**

Retourne the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state B before automatically switching back in to A state.

```
js function get_maxTimeOnStateB( )
nodejs function get_maxTimeOnStateB( )
php function get_maxTimeOnStateB( )
cpp s64 get_maxTimeOnStateB( )
m -(s64) maxTimeOnStateB
pas function get_maxTimeOnStateB( ): int64
vb function get_maxTimeOnStateB( ) As Long
cs long get_maxTimeOnStateB( )
java long get_maxTimeOnStateB( )
py def get_maxTimeOnStateB( )
cmd YWatchdog target get_maxTimeOnStateB
```

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 online.

**Returns :**

an instance of YModule

**watchdog→get\_module\_async()**  
**watchdog→module\_async()****YWatchdog**

Gets the `YModule` object for the device on which the function is located (asynchronous version).

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

**Parameters :**

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

**Returns :**

nothing : the result is provided to the callback.

**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) <b>output</b>
pas	function <b>get_output( ): Integer</b>
vb	function <b>get_output( ) As Integer</b>
cs	int <b>get_output( )</b>
java	int <b>get_output( )</b>
py	def <b>get_output( )</b>
cmd	YWatchdog <b>target 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()**  
**watchdog→pulseTimer()watchdog→**  
**get\_pulseTimer( )**

**YWatchdog**

Returns the number of milliseconds remaining before the watchdog is returned to idle position (state A), during a measured pulse generation.

```
js function get_pulseTimer( )
nodejs function get_pulseTimer( )
php function get_pulseTimer( )
cpp s64 get_pulseTimer( )
m -(s64) pulseTimer
pas function get_pulseTimer( ): int64
vb function get_pulseTimer( ) As Long
cs long get_pulseTimer( )
java long get_pulseTimer( )
py def get_pulseTimer( )
cmd YWatchdog target get_pulseTimer
```

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\_PULSE\_TIMER\_INVALID.

**watchdog→get\_running()****YWatchdog****watchdog→running()watchdog→get\_running( )**

Returns the watchdog running state.

```
js function get_running( )
nodejs function get_running( )
php function get_running( )
cpp Y_RUNNING_enum get_running( )
m -(Y_RUNNING_enum) running
pas function get_running( ): Integer
vb function get_running( ) As Integer
cs int get_running( )
java int get_running( )
py def get_running( )
cmd YWatchdog target get_running
```

**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>
node.js	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	<b>YWatchdog target 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()**  
**watchdog→stateAtPowerOn()** watchdog→  
**get\_stateAtPowerOn( )**

**YWatchdog**

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 get_stateAtPowerOn( )
nodejs function get_stateAtPowerOn( )
php function get_stateAtPowerOn( )
cpp Y_STATEATPOWERON_enum get_stateAtPowerOn( )
m -(Y_STATEATPOWERON_enum) stateAtPowerOn
pas function get_stateAtPowerOn( ): Integer
vb function get_stateAtPowerOn( ) As Integer
cs int get_stateAtPowerOn( )
java int get_stateAtPowerOn( )
py def get_stateAtPowerOn( )
cmd YWatchdog target get_stateAtPowerOn
```

**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()**  
**watchdog→triggerDelay()** watchdog→  
**get\_triggerDelay()**

**YWatchdog**

Returns the waiting duration before a reset is automatically triggered by the watchdog, in milliseconds.

```
js function get_triggerDelay( )  
nodejs function get_triggerDelay( )  
php function get_triggerDelay( )  
cpp s64 get_triggerDelay( )  
m -(s64) triggerDelay  
pas function get_triggerDelay( ): int64  
vb function get_triggerDelay( ) As Long  
cs long get_triggerDelay( )  
java long get_triggerDelay( )  
py def get_triggerDelay( )  
cmd YWatchdog target get_triggerDelay
```

**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()**  
**watchdog→triggerDuration()watchdog→**  
**get\_triggerDuration( )**

**YWatchdog**

Returns the duration of resets caused by the watchdog, in milliseconds.

<b>js</b>	function <b>get_triggerDuration( )</b>
<b>nodejs</b>	function <b>get_triggerDuration( )</b>
<b>php</b>	function <b>get_triggerDuration( )</b>
<b>cpp</b>	s64 <b>get_triggerDuration( )</b>
<b>m</b>	-(s64) triggerDuration
<b>pas</b>	function <b>get_triggerDuration( )</b> : int64
<b>vb</b>	function <b>get_triggerDuration( )</b> As Long
<b>cs</b>	long <b>get_triggerDuration( )</b>
<b>java</b>	long <b>get_triggerDuration( )</b>
<b>py</b>	def <b>get_triggerDuration( )</b>
<b>cmd</b>	<b>YWatchdog target 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 get(userData)
node.js	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): Tobject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

**Returns :**

the object stored previously by the caller.

**watchdog→isOnline()****YWatchdog**

Checks if the watchdog is currently reachable, without raising any error.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) 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	<b>def 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()****YWatchdog**

Preloads the watchdog cache with a specified validity duration.

js	function <b>load( msValidity)</b>
node.js	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( 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>
cpp	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()****YWatchdog**

Sets the relay to output B (active) for a specified duration, then brings it automatically back to output A (idle state).

js	function pulse( ms_duration)
nodejs	function pulse( ms_duration)
php	function pulse( \$ms_duration)
cpp	int pulse( int ms_duration)
m	-(int) pulse : (int) ms_duration
pas	function pulse( ms_duration: LongInt): integer
vb	function pulse( ByVal ms_duration As Integer) As Integer
cs	int pulse( int ms_duration)
java	int pulse( int ms_duration)
py	def pulse( ms_duration)
cmd	YWatchdog target pulse ms_duration

**Parameters :**

**ms\_duration** pulse duration, in millisecondes

**Returns :**

YAPI\_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

## **watchdog→registerValueCallback()**

**YWatchdog**

### **registerValueCallback( )**

Registers the callback function that is invoked on every change of advertised value.

<b>js</b>	<b>function registerValueCallback( callback)</b>
<b>node.js</b>	<b>function registerValueCallback( callback)</b>
<b>php</b>	<b>function registerValueCallback( \$callback)</b>
<b>cpp</b>	<b>int registerValueCallback( YWatchdogValueCallback callback)</b>
<b>m</b>	<b>-(int) registerValueCallback : (YWatchdogValueCallback) callback</b>
<b>pas</b>	<b>function registerValueCallback( callback: TYWatchdogValueCallback): LongInt</b>
<b>vb</b>	<b>function registerValueCallback( ) As Integer</b>
<b>cs</b>	<b>int registerValueCallback( ValueCallback callback)</b>
<b>java</b>	<b>int registerValueCallback( UpdateCallback callback)</b>
<b>py</b>	<b>def registerValueCallback( 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>
node.js	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	<b>YWatchdog target 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()**  
**watchdog→setAutoStart()**  
**watchdog→set\_autoStart( )**

**YWatchdog**

Changes the watchdog runningsttae at module power up.

js	function <b>set_autoStart( newval)</b>
node.js	function <b>set_autoStart( newval)</b>
php	function <b>set_autoStart( \$newval)</b>
cpp	int <b>set_autoStart( Y_AUTOSTART_enum newval)</b>
m	-(int) <b>setAutoStart : (Y_AUTOSTART_enum) newval</b>
pas	function <b>set_autoStart( newval: Integer): integer</b>
vb	function <b>set_autoStart( ByVal newval As Integer) As Integer</b>
cs	int <b>set_autoStart( int newval)</b>
java	int <b>set_autoStart( int newval)</b>
py	def <b>set_autoStart( newval)</b>
cmd	<b>YWatchdog 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()**  
**watchdog→setLogicalName()** watchdog→  
**set\_logicalName( )**

**YWatchdog**

Changes the logical name of the watchdog.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YWatchdog target set_logicalName newval
```

You can use `yCheckLogicalName( )` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash( )` method of the module if the modification must be kept.

**Parameters :**

**newval** a string corresponding to the logical name of the watchdog.

**Returns :**

`YAPI_SUCCESS` if the call succeeds. On failure, throws an exception or returns a negative error code.

**watchdog→set\_maxTimeOnStateA()**  
**watchdog→setMaxTimeOnStateA()**  
**watchdog→set\_maxTimeOnStateA( )**

**YWatchdog**

Sets the maximum time (ms) allowed for \$THEFUNCTIONS\$ to stay in state A before automatically switching back in to B state.

<b>js</b>	function <b>set_maxTimeOnStateA( newval)</b>
<b>nodejs</b>	function <b>set_maxTimeOnStateA( newval)</b>
<b>php</b>	function <b>set_maxTimeOnStateA( \$newval)</b>
<b>cpp</b>	int <b>set_maxTimeOnStateA( s64 newval)</b>
<b>m</b>	-(int) <b>setMaxTimeOnStateA : (s64) newval</b>
<b>pas</b>	function <b>set_maxTimeOnStateA( newval: int64): integer</b>
<b>vb</b>	function <b>set_maxTimeOnStateA( ByVal newval As Long) As Integer</b>
<b>cs</b>	int <b>set_maxTimeOnStateA( long newval)</b>
<b>java</b>	int <b>set_maxTimeOnStateA( long newval)</b>
<b>py</b>	def <b>set_maxTimeOnStateA( newval)</b>
<b>cmd</b>	<b>YWatchdog target set_maxTimeOnStateA 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()**  
**watchdog→setMaxTimeOnStateB()** watchdog→  
**set\_maxTimeOnStateB( )**

**YWatchdog**

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( newval)</b>
nodejs	function <b>set_maxTimeOnStateB( newval)</b>
php	function <b>set_maxTimeOnStateB( \$newval)</b>
cpp	int <b>set_maxTimeOnStateB( s64 newval)</b>
m	- (int) <b>setMaxTimeOnStateB : (s64) newval</b>
pas	function <b>set_maxTimeOnStateB( newval: int64): integer</b>
vb	function <b>set_maxTimeOnStateB( ByVal newval As Long) As Integer</b>
cs	int <b>set_maxTimeOnStateB( long newval)</b>
java	int <b>set_maxTimeOnStateB( long newval)</b>
py	def <b>set_maxTimeOnStateB( newval)</b>
cmd	<b>YWatchdog 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).

<code>js</code>	<code>function set_output( newval)</code>
<code>nodejs</code>	<code>function set_output( newval)</code>
<code>php</code>	<code>function set_output( \$newval)</code>
<code>cpp</code>	<code>int set_output( Y_OUTPUT_enum newval)</code>
<code>m</code>	<code>-(int) setOutput : (Y_OUTPUT_enum) newval</code>
<code>pas</code>	<code>function set_output( newval: Integer): integer</code>
<code>vb</code>	<code>function set_output( ByVal newval As Integer) As Integer</code>
<code>cs</code>	<code>int set_output( int newval)</code>
<code>java</code>	<code>int set_output( int newval)</code>
<code>py</code>	<code>def set_output( newval)</code>
<code>cmd</code>	<code>YWatchdog target set_output newval</code>

**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()**  
**watchdog→setRunning()**  
**watchdog→set\_running( )****YWatchdog**

Changes the running state of the watchdog.

js	function <b>set_running( newval)</b>
nodejs	function <b>set_running( newval)</b>
php	function <b>set_running( \$newval)</b>
cpp	int <b>set_running( Y_RUNNING_enum newval)</b>
m	-(int) <b>setRunning : (Y_RUNNING_enum) newval</b>
pas	function <b>set_running( newval: Integer): integer</b>
vb	function <b>set_running( ByVal newval As Integer) As Integer</b>
cs	int <b>set_running( int newval)</b>
java	int <b>set_running( int newval)</b>
py	def <b>set_running( newval)</b>
cmd	<b>YWatchdog 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).

<b>js</b>	function <b>set_state( newval)</b>
<b>nodejs</b>	function <b>set_state( newval)</b>
<b>php</b>	function <b>set_state( \$newval)</b>
<b>cpp</b>	int <b>set_state( Y_STATE_enum newval)</b>
<b>m</b>	-(int) <b>setState : (Y_STATE_enum) newval</b>
<b>pas</b>	function <b>set_state( newval: Integer): integer</b>
<b>vb</b>	function <b>set_state( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_state( int newval)</b>
<b>java</b>	int <b>set_state( int newval)</b>
<b>py</b>	def <b>set_state( newval)</b>
<b>cmd</b>	<b>YWatchdog 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 set_stateAtPowerOn( newval)
nodejs function set_stateAtPowerOn( newval)
php  function set_stateAtPowerOn( $newval)
cpp   int set_stateAtPowerOn( Y_STATEATPOWERON_enum newval)
m    -(int) setStateAtPowerOn : (Y_STATEATPOWERON_enum) newval
pas   function set_stateAtPowerOn( newval: Integer): integer
vb    function set_stateAtPowerOn( ByVal newval As Integer) As Integer
cs    int set_stateAtPowerOn( int newval)
java  int set_stateAtPowerOn( int newval)
py    def set_stateAtPowerOn( newval)
cmd   YWatchdog target set_stateAtPowerOn newval

```

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.

<b>js</b>	function <b>set_triggerDelay( newval)</b>
<b>nodejs</b>	function <b>set_triggerDelay( newval)</b>
<b>php</b>	function <b>set_triggerDelay( \$newval)</b>
<b>cpp</b>	int <b>set_triggerDelay( s64 newval)</b>
<b>m</b>	-(int) <b>setTriggerDelay : (s64) newval</b>
<b>pas</b>	function <b>set_triggerDelay( newval: int64): integer</b>
<b>vb</b>	function <b>set_triggerDelay( ByVal newval As Long) As Integer</b>
<b>cs</b>	int <b>set_triggerDelay( long newval)</b>
<b>java</b>	int <b>set_triggerDelay( long newval)</b>
<b>py</b>	def <b>set_triggerDelay( newval)</b>
<b>cmd</b>	<b>YWatchdog target set_triggerDelay 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()**  
**watchdog→setTriggerDuration()**  
**watchdog→set\_triggerDuration( )**

**YWatchdog**

Changes the duration of resets caused by the watchdog, in milliseconds.

```
js function set_triggerDuration( newval)
nodejs function set_triggerDuration( newval)
php function set_triggerDuration( $newval)
cpp int set_triggerDuration( s64 newval)
m -(int) setTriggerDuration : (s64) newval
pas function set_triggerDuration( newval: int64): integer
vb function set_triggerDuration( ByVal newval As Long) As Integer
cs int set_triggerDuration( long newval)
java int set_triggerDuration( long newval)
py def set_triggerDuration( newval)
cmd YWatchdog target set_triggerDuration newval
```

**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)**  
**watchdog→setUserData()watchdog→**  
**set(userData( )**

**YWatchdog**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData( data)</b>
node.js	function <b>set(userData( data)</b>
php	function <b>set(userData( \$data)</b>
cpp	void <b>set(userData( void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData( data: Tobject)</b>
vb	procedure <b>set(userData( ByVal data As Object)</b>
cs	void <b>set(userData( object data)</b>
java	void <b>set(userData( Object data)</b>
py	def <b>set(userData( 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');
cpp #include "yocto_wireless.h"
m #import "yocto_wireless.h"
pas uses yocto_wireless;
vb yocto_wireless.vb
cs yocto_wireless.cs
java import com.yoctopuce.YoctoAPI.YWireless;
py from yocto_wireless import *

```

### Global functions

#### **yFindWireless(func)**

Retrieves a wireless lan interface for a given identifier.

#### **yFirstWireless()**

Starts the enumeration of wireless lan interfaces currently accessible.

### YWireless methods

#### **wireless→adhocNetwork(ssid, securityKey)**

Changes the configuration of the wireless lan interface to create an ad-hoc wireless network, without using an access point.

#### **wireless→describe()**

Returns a short text that describes unambiguously the instance of the wireless lan interface in the form TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **wireless→get\_advertisedValue()**

Returns the current value of the wireless lan interface (no more than 6 characters).

#### **wireless→get\_channel()**

Returns the 802.11 channel currently used, or 0 when the selected network has not been found.

#### **wireless→get\_detectedWlans()**

Returns a list of YWlanRecord objects that describe detected Wireless networks.

#### **wireless→get\_errorMessage()**

Returns the error message of the latest error with the wireless lan interface.

#### **wireless→get\_errorType()**

Returns the numerical error code of the latest error with the wireless lan interface.

#### **wireless→get\_friendlyName()**

Returns a global identifier of the wireless lan interface in the format MODULE\_NAME . FUNCTION\_NAME.

#### **wireless→get\_functionDescriptor()**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

#### **wireless→get\_functionId()**

Returns the hardware identifier of the wireless lan interface, without reference to the module.

#### **wireless→get\_hardwareId()**

Returns the unique hardware identifier of the wireless lan interface in the form SERIAL . FUNCTIONID.

### 3. Reference

#### 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()****YWireless****yFindWireless()yFindWireless( )**

Retrieves a wireless lan interface for a given identifier.

<b>js</b>	<code>function yFindWireless( func)</code>
<b>nodejs</b>	<code>function FindWireless( func)</code>
<b>php</b>	<code>function yFindWireless( \$func)</code>
<b>cpp</b>	<code>YWireless* yFindWireless( string func)</code>
<b>m</b>	<code>+ (YWireless*) yFindWireless : (NSString*) func</code>
<b>pas</b>	<code>function yFindWireless( func: string): TYWireless</code>
<b>vb</b>	<code>function yFindWireless( ByVal func As String) As YWireless</code>
<b>cs</b>	<code>YWireless FindWireless( string func)</code>
<b>java</b>	<code>YWireless FindWireless( String func)</code>
<b>py</b>	<code>def FindWireless( 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 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()yFirstWireless()

**YWireless**

Starts the enumeration of wireless lan interfaces currently accessible.

js	function <b>yFirstWireless( )</b>
node.js	function <b>FirstWireless( )</b>
php	function <b>yFirstWireless( )</b>
cpp	YWireless* <b>yFirstWireless( )</b>
m	YWireless* <b>yFirstWireless( )</b>
pas	function <b>yFirstWireless( ): TYWireless</b>
vb	function <b>yFirstWireless( ) As YWireless</b>
cs	<b>YWireless FirstWireless( )</b>
java	<b>YWireless 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	- <b>(int) adhocNetwork : (NSString*) ssid</b> <b>: (NSString*) 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 target <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)=RELAYL01-123456.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)=RELAYL01-123456.relay1)

**wireless→get\_advertisedValue()**  
**wireless→advertisedValue()wireless→**  
**get\_advertisedValue( )**

**YWireless**

Returns the current value of the wireless lan interface (no more than 6 characters).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	<b>YWireless target 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>
node.js	function <b>get_channel( )</b>
php	function <b>get_channel( )</b>
cpp	int <b>get_channel( )</b>
m	-(int) <b>channel</b>
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	<b>YWireless target 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()**  
**wireless→detectedWlans()wireless→**  
**get\_detectedWlans()**

**YWireless**

Returns a list of YWlanRecord objects that describe detected Wireless networks.

<code>js</code>	<code>function get_detectedWlans( )</code>
<code>node.js</code>	<code>function get_detectedWlans( )</code>
<code>php</code>	<code>function get_detectedWlans( )</code>
<code>cpp</code>	<code>vector&lt;YWlanRecord&gt; get_detectedWlans( )</code>
<code>m</code>	<code>-(NSMutableArray*) detectedWlans</code>
<code>pas</code>	<code>function get_detectedWlans( ): TYWlanRecordArray</code>
<code>vb</code>	<code>function get_detectedWlans( ) As List</code>
<code>cs</code>	<code>List&lt;YWlanRecord&gt; get_detectedWlans( )</code>
<code>java</code>	<code>ArrayList&lt;YWlanRecord&gt; get_detectedWlans( )</code>
<code>py</code>	<code>def get_detectedWlans( )</code>
<code>cmd</code>	<code>YWireless target get_detectedWlans</code>

This list is not updated when the module is already connected to an acces point (infrastructure mode). To force an update of this list, `adhocNetwork( )` must be called to disconnect the module from the current network. The returned list must be unallocated by the caller.

**Returns :**

a list of YWlanRecord objects, containing the SSID, channel, link quality and the type of security of the wireless network.

On failure, throws an exception or returns an empty list.

**wireless→get\_errorMessage()**  
**wireless→errorMessage()wireless→**  
**get\_errorMessage( )**

**YWireless**

Returns the error message of the latest error with the wireless lan interface.

js	function get_errorMessage( )
nodejs	function get_errorMessage( )
php	function get_errorMessage( )
cpp	string get_errorMessage( )
m	-(NSString*) errorMessage
pas	function get_errorMessage( ): string
vb	function get_errorMessage( ) As String
cs	string get_errorMessage( )
java	String get_errorMessage( )
py	def get_errorMessage( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

**Returns :**

a string corresponding to the latest error message that 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()  
wireless→friendlyName()wireless→  
get\_friendlyName()

YWireless

Returns a global identifier of the wireless lan interface in the format MODULE\_NAME . FUNCTION\_NAME.

```
js function get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

The returned string uses the logical names of the module and of the wireless lan interface if they are defined, otherwise the serial number of the module and the hardware identifier of the wireless lan interface (for exemple: MyCustomName.relay1)

**Returns :**

a string that uniquely identifies the wireless lan interface using logical names (ex: MyCustomName.relay1) On failure, throws an exception or returns Y\_FRIENDLYNAME\_INVALID.

**wireless→get\_functionDescriptor()**  
**wireless→functionDescriptor()wireless→**  
**get\_functionDescriptor( )**

**YWireless**

Returns a unique identifier of type YFUN\_DESCR corresponding to the function.

js	function <b>get_functionDescriptor( )</b>
node.js	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.

**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()**  
**wireless→hardwareId()wireless→**  
**get\_hardwareId( )**

**YWireless**

Returns the unique hardware identifier of the wireless lan interface in the form SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.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( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	<b>def 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 RELAYL01-123456.relay1)

**Returns :**

a string that uniquely identifies the wireless lan interface (ex: RELAYL01-123456.relay1) On failure, throws an exception or returns Y\_HARDWAREID\_INVALID.

wireless→get\_linkQuality()  
wireless→linkQuality()wireless→  
get\_linkQuality()

YWireless

Returns the link quality, expressed in percent.

js	function get_linkQuality( )
nodejs	function get_linkQuality( )
php	function get_linkQuality( )
cpp	int get_linkQuality( )
m	-(int) linkQuality
pas	function get_linkQuality( ): LongInt
vb	function get_linkQuality( ) As Integer
cs	int get_linkQuality( )
java	int get_linkQuality( )
py	def get_linkQuality( )
cmd	YWireless target get_linkQuality

**Returns :**

an integer corresponding to the link quality, expressed in percent

On failure, throws an exception or returns Y\_LINKQUALITY\_INVALID.

**wireless→get\_logicalName()**  
**wireless→logicalName()wireless→**  
**get\_logicalName( )**

**YWireless**

Returns the logical name of the wireless lan interface.

<b>js</b>	function <b>get_logicalName( )</b>
<b>nodejs</b>	function <b>get_logicalName( )</b>
<b>php</b>	function <b>get_logicalName( )</b>
<b>cpp</b>	string <b>get_logicalName( )</b>
<b>m</b>	-(NSString*) logicalName
<b>pas</b>	function <b>get_logicalName( )</b> : string
<b>vb</b>	function <b>get_logicalName( )</b> As String
<b>cs</b>	string <b>get_logicalName( )</b>
<b>java</b>	String <b>get_logicalName( )</b>
<b>py</b>	def <b>get_logicalName( )</b>
<b>cmd</b>	YWireless <b>target 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>
node.js	function <b>get_message()</b>
php	function <b>get_message()</b>
cpp	string <b>get_message()</b>
m	-(NSString*) <b>message</b>
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	<b>YWireless target 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>
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 online.

**Returns :**

an instance of YModule

## wireless→get\_module\_async() wireless→module\_async()

YWireless

Gets the `YModule` object for the device on which the function is located (asynchronous version).

```
js  function get_module_async( callback, context )
node.js function get_module_async( callback, context )
```

If the function cannot be located on any module, the returned `YModule` object does not show as online. This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

### Parameters :

**callback** callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested `YModule` object

**context** caller-specific object that is passed as-is to the callback function

### Returns :

nothing : the result is provided to the callback.

**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) <b>security</b>
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 target <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 get_ssid( )
node.js function get_ssid( )
php function get_ssid( )
cpp string get_ssid( )
m -(NSString*) ssid
pas function get_ssid( ): string
vb function get_ssid( ) As String
cs string get_ssid( )
java String get_ssid( )
py def get_ssid( )
cmd YWireless target get_ssid
```

**Returns :**

a string corresponding to the wireless network name (SSID)

On failure, throws an exception or returns Y\_SSID\_INVALID.

**wireless→get(userData)****YWireless****wireless→userData()wireless→get(userData()**

Returns the value of the userData attribute, as previously stored using method `set(userData)`.

js	<code>function get(userData) </code>
nodejs	<code>function get(userData) </code>
php	<code>function get(userData) </code>
cpp	<code>void * get(userData) </code>
m	<code>-(void*) userData </code>
pas	<code>function get(userData): Tobject </code>
vb	<code>function get(userData) As Object </code>
cs	<code>object get(userData) </code>
java	<code>Object get(userData) </code>
py	<code>def get(userData) </code>

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	- <b>(BOOL) 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	<b>def 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 joinNetwork( ssid, securityKey)
nodejs function joinNetwork( ssid, securityKey)
php function joinNetwork( $ssid, $securityKey)
cpp int joinNetwork( string ssid, string securityKey)
m -(int) joinNetwork : (NSString*) ssid
                  : (NSString*) securityKey
pas function joinNetwork( ssid: string, securityKey: string): integer
vb function joinNetwork( ByVal ssid As String,
                        ByVal securityKey As String) As Integer
cs int joinNetwork( string ssid, string securityKey)
java int joinNetwork( String ssid, String securityKey)
py def joinNetwork( ssid, securityKey)
cmd YWireless target joinNetwork ssid securityKey
```

Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

**Parameters :**

**ssid** the name of the network to connect to  
**securityKey** the network key, as a character string

**Returns :**

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

**wireless→load()wireless→load( )****YWireless**

Preloads the wireless lan interface 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.

## 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 registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( \$callback)
cpp	int registerValueCallback( YWirelessValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YWirelessValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYWirelessValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <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()**  
**wireless→setLogicalName()** wireless→  
**set\_logicalName( )**

**YWireless**

Changes the logical name of the wireless lan interface.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YWireless 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)**  
**wireless→setUserData(wireless→**  
**set(userData))**

**YWireless**

Stores a user context provided as argument in the userData attribute of the function.

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b>
m	-(void) <b>set(userData)</b>
pas	procedure <b>set(userData)</b>
vb	procedure <b>set(userData)</b>
cs	void <b>set(userData)</b>
java	void <b>set(userData)</b>
py	def <b>set(userData)</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 38  
adhocNetwork, YWireless 1730  
AnButton 84

## B

Blueprint 12

## C

C++ 3, 8  
calibrate, YLightSensor 767  
calibrateFromPoints, YAccelerometer 42  
calibrateFromPoints, YCarbonDioxide 130  
calibrateFromPoints, YCompass 206  
calibrateFromPoints, YCurrent 250  
calibrateFromPoints, YGenericSensor 559  
calibrateFromPoints, YGyro 609  
calibrateFromPoints, YHumidity 693  
calibrateFromPoints, YLightSensor 768  
calibrateFromPoints, YMagnetometer 811  
calibrateFromPoints, YPower 997  
calibrateFromPoints, YPressure 1044  
calibrateFromPoints, YQt 1156  
calibrateFromPoints,YSensor 1310  
calibrateFromPoints, YTemperature 1392  
calibrateFromPoints, YTilt 1437  
calibrateFromPoints, YVoc 1480  
calibrateFromPoints, YVoltage 1523  
callbackLogin, YNetwork 910  
cancel3DCalibration, YRefFrame 1230  
CarbonDioxide 126  
CheckLogicalName, YAPI 14  
clear, YDisplayLayer 462  
clearConsole, YDisplayLayer 463  
Clock 1195  
ColorLed 169  
Compass 202  
Configuration 1226  
consoleOut, YDisplayLayer 464  
copyLayerContent, YDisplay 414  
Current 246

## D

Data 324, 334, 347  
DataLogger 289  
delayedPulse, YDigitalIO 366  
delayedPulse, YRelay 1270  
delayedPulse, YWatchdog 1682  
describe, YAccelerometer 43  
describe, YAnButton 88  
describe, YCarbonDioxide 131  
describe, YColorLed 172

describe, YCompass 207  
describe, YCurrent 251  
describe, YDataLogger 292  
describe, YDigitalIO 367  
describe, YDisplay 415  
describe, YDualPower 496  
describe, YFiles 525  
describe, YGenericSensor 560  
describe, YGyro 610  
describe, YHubPort 663  
describe, YHumidity 694  
describe, YLed 735  
describe, YLightSensor 769  
describe, YMagnetometer 812  
describe, YModule 863  
describe, YNetwork 911  
describe, YOsControl 969  
describe, YPower 998  
describe, YPressure 1045  
describe, YPwmOutput 1087  
describe, YPwmPowerSource 1128  
describe, YQt 1157  
describe, YRealTimeClock 1198  
describe, YRefFrame 1231  
describe, YRelay 1271  
describe, YSensor 1311  
describe, YServo 1353  
describe, YTemperature 1393  
describe, YTilt 1438  
describe, YVoc 1481  
describe, YVoltage 1524  
describe, YVSource 1565  
describe, YWakeUpMonitor 1602  
describe, YWakeUpSchedule 1641  
describe, YWatchdog 1683  
describe, YWireless 1731  
Digital 362  
DisableExceptions, YAPI 15  
Display 410  
DisplayLayer 461  
download, YFiles 526  
download, YModule 864  
download\_async, YFiles 527  
drawBar, YDisplayLayer 465  
drawBitmap, YDisplayLayer 466  
drawCircle, YDisplayLayer 467  
drawDisc, YDisplayLayer 468  
drawImage, YDisplayLayer 469  
drawPixel, YDisplayLayer 470  
drawRect, YDisplayLayer 471  
drawText, YDisplayLayer 472  
dutyCycleMove, YPwmOutput 1088

## E

EnableExceptions, YAPI 16  
EnableUSBHost, YAPI 17  
Error 7  
External 493

## F

fade, YDisplay 416  
Files 522  
FindAccelerometer, YAccelerometer 40  
FindAnButton, YAnButton 86  
FindCarbonDioxide, YCarbonDioxide 128  
FindColorLed, YColorLed 170  
FindCompass, YCompass 204  
FindCurrent, YCurrent 248  
FindDataLogger, YDataLogger 290  
FindDigitalIO, YDigitalIO 364  
FindDisplay, YDisplay 412  
FindDualPower, YDualPower 494  
FindFiles, YFiles 523  
FindGenericSensor, YGenericSensor 557  
FindGyro, YGyro 607  
FindHubPort, YHubPort 661  
FindHumidity, YHumidity 691  
FindLed, YLed 733  
FindLightSensor, YLightSensor 765  
FindMagnetometer, YMagnetometer 809  
FindModule, YModule 861  
FindNetwork, YNetwork 908  
FindOsControl, YOsControl 967  
FindPower, YPower 995  
FindPressure, YPressure 1042  
FindPwmOutput, YPwmOutput 1085  
FindPwmPowerSource, YPwmPowerSource 1126  
FindQt, YQt 1154  
FindRealTimeClock, YRealTimeClock 1196  
FindRefFrame, YRefFrame 1228  
FindRelay, YRelay 1268  
FindSensor, YSensor 1308  
FindServo,YServo 1351  
FindTemperature, YTemperature 1390  
FindTilt, YTilt 1435  
FindVoc, YVoc 1478  
FindVoltage, YVoltage 1521  
FindVSource, YVSource 1563  
FindWakeUpMonitor, YWakeUpMonitor 1600  
FindWakeUpSchedule, YWakeUpSchedule 1639  
FindWatchdog, YWatchdog 1680  
FindWireless, YWireless 1728  
FirstAccelerometer, YAccelerometer 41  
FirstAnButton, YAnButton 87  
FirstCarbonDioxide, YCarbonDioxide 129  
FirstColorLed, YColorLed 171  
FirstCompass, YCompass 205  
FirstCurrent, YCurrent 249  
FirstDataLogger, YDataLogger 291  
FirstDigitalIO, YDigitalIO 365

FirstDisplay, YDisplay 413  
FirstDualPower, YDualPower 495  
FirstFiles, YFiles 524  
FirstGenericSensor, YGenericSensor 558  
FirstGyro, YGyro 608  
FirstHubPort, YHubPort 662  
FirstHumidity, YHumidity 692  
FirstLed, YLed 734  
FirstLightSensor, YLightSensor 766  
FirstMagnetometer, YMagnetometer 810  
FirstModule, YModule 862  
FirstNetwork, YNetwork 909  
FirstOsControl, YOsControl 968  
FirstPower, YPower 996  
FirstPressure, YPressure 1043  
FirstPwmOutput, YPwmOutput 1086  
FirstPwmPowerSource, YPwmPowerSource 1127  
FirstQt, YQt 1155  
FirstRealTimeClock, YRealTimeClock 1197  
FirstRefFrame, YRefFrame 1229  
FirstRelay, YRelay 1269  
FirstSensor, YSensor 1309  
FirstServo, YServo 1352  
FirstTemperature, YTemperature 1391  
FirstTilt, YTilt 1436  
FirstVoc, YVoc 1479  
FirstVoltage, YVoltage 1522  
FirstVSource, YVSource 1564  
FirstWakeUpMonitor, YWakeUpMonitor 1601  
FirstWakeUpSchedule, YWakeUpSchedule 1640  
FirstWatchdog, YWatchdog 1681  
FirstWireless, YWireless 1729  
forgetAllDataStreams, YDataLogger 293  
format\_fs, YFiles 528  
Formatted 324  
Frame 1226  
FreeAPI, YAPI 18  
functionCount, YModule 865  
functionId, YModule 866  
functionName, YModule 867  
Functions 13  
functionValue, YModule 868

## G

General 13  
GenericSensor 555  
get\_3DCalibrationHint, YRefFrame 1232  
get\_3DCalibrationLogMsg, YRefFrame 1233  
get\_3DCalibrationProgress, YRefFrame 1234  
get\_3DCalibrationStage, YRefFrame 1235  
get\_3DCalibrationStageProgress, YRefFrame 1236  
get\_adminPassword, YNetwork 912  
get\_advertisedValue, YAccelerometer 44  
get\_advertisedValue, YAnButton 89  
get\_advertisedValue, YCarbonDioxide 132  
get\_advertisedValue, YColorLed 173  
get\_advertisedValue, YCompass 208

get\_advertisedValue, YCurrent 252  
get\_advertisedValue, YDataLogger 294  
get\_advertisedValue, YDigitalIO 368  
get\_advertisedValue, YDisplay 417  
get\_advertisedValue, YDualPower 497  
get\_advertisedValue, YFiles 529  
get\_advertisedValue, YGenericSensor 561  
get\_advertisedValue, YGyro 611  
get\_advertisedValue, YHubPort 664  
get\_advertisedValue, YHumidity 695  
get\_advertisedValue, YLed 736  
get\_advertisedValue, YLightSensor 770  
get\_advertisedValue, YMagnetometer 813  
get\_advertisedValue, YNetwork 913  
get\_advertisedValue, YOsControl 970  
get\_advertisedValue, YPower 999  
get\_advertisedValue, YPressure 1046  
get\_advertisedValue, YPwmOutput 1089  
get\_advertisedValue, YPwmPowerSource 1129  
get\_advertisedValue, YQt 1158  
get\_advertisedValue, YRealTimeClock 1199  
get\_advertisedValue, YRefFrame 1237  
get\_advertisedValue, YRelay 1272  
get\_advertisedValue, YSensor 1312  
get\_advertisedValue, YServo 1354  
get\_advertisedValue, YTemperature 1394  
get\_advertisedValue, YTilt 1439  
get\_advertisedValue, YVoc 1482  
get\_advertisedValue, YVoltage 1525  
get\_advertisedValue, YVSource 1566  
get\_advertisedValue, YWakeUpMonitor 1603  
get\_advertisedValue, YWakeUpSchedule 1642  
get\_advertisedValue, YWatchdog 1684  
get\_advertisedValue, YWireless 1732  
get\_analogCalibration, YAnButton 90  
get\_autoStart, YDataLogger 295  
get\_autoStart, YWatchdog 1685  
get\_averageValue, YDataRun 324  
get\_averageValue, YDataStream 348  
get\_averageValue, YMeasure 853  
get\_baudRate, YHubPort 665  
get\_beacon, YModule 869  
get\_bearing, YRefFrame 1238  
get\_bitDirection, YDigitalIO 369  
get\_bitOpenDrain, YDigitalIO 370  
get\_bitPolarity, YDigitalIO 371  
get\_bitState, YDigitalIO 372  
get\_blinking, YLed 737  
get\_brightness, YDisplay 418  
get\_calibratedValue, YAnButton 91  
get\_calibrationMax, YAnButton 92  
get\_calibrationMin, YAnButton 93  
get\_callbackCredentials, YNetwork 914  
get\_callbackEncoding, YNetwork 915  
get\_callbackMaxDelay, YNetwork 916  
get\_callbackMethod, YNetwork 917  
get\_callbackMinDelay, YNetwork 918  
get\_callbackUrl, YNetwork 919  
get\_channel, YWireless 1733  
get\_columnCount, YDataStream 349  
get\_columnNames, YDataStream 350  
get\_cosPhi, YPower 1000  
get\_countdown, YRelay 1273  
get\_countdown, YWatchdog 1686  
get\_currentRawValue, YAccelerometer 45  
get\_currentRawValue, YCarbonDioxide 133  
get\_currentRawValue, YCompass 209  
get\_currentRawValue, YCurrent 253  
get\_currentRawValue, YGenericSensor 562  
get\_currentRawValue, YGyro 612  
get\_currentRawValue, YHumidity 696  
get\_currentRawValue, YLightSensor 771  
get\_currentRawValue, YMagnetometer 814  
get\_currentRawValue, YPower 1001  
get\_currentRawValue, YPressure 1047  
get\_currentRawValue, YQt 1159  
get\_currentRawValue, YSensor 1313  
get\_currentRawValue, YTemperature 1395  
get\_currentRawValue, YTilt 1440  
get\_currentRawValue, YVoc 1483  
get\_currentRawValue, YVoltage 1526  
get\_currentRunIndex, YDataLogger 296  
get\_currentValue, YAccelerometer 46  
get\_currentValue, YCarbonDioxide 134  
get\_currentValue, YCompass 210  
get\_currentValue, YCurrent 254  
get\_currentValue, YGenericSensor 563  
get\_currentValue, YGyro 613  
get\_currentValue, YHumidity 697  
get\_currentValue, YLightSensor 772  
get\_currentValue, YMagnetometer 815  
get\_currentValue, YPower 1002  
get\_currentValue, YPressure 1048  
get\_currentValue, YQt 1160  
get\_currentValue, YSensor 1314  
get\_currentValue, YTemperature 1396  
get\_currentValue, YTilt 1441  
get\_currentValue, YVoc 1484  
get\_currentValue, YVoltage 1527  
get\_data, YDataStream 351  
get\_dataRows, YDataStream 352  
get\_dataSamplesIntervalMs, YDataStream 353  
get\_dataSets, YDataLogger 297  
get\_dataStreams, YDataLogger 298  
get\_dateTime, YRealTimeClock 1200  
get\_detectedWlans, YWireless 1734  
get\_discoverable, YNetwork 920  
get\_display, YDisplayLayer 473  
get\_displayHeight, YDisplay 419  
get\_displayHeight, YDisplayLayer 474  
get\_displayLayer, YDisplay 420  
get\_displayType, YDisplay 421  
get\_displayWidth, YDisplay 422  
get\_displayWidth, YDisplayLayer 475  
get\_duration, YDataRun 325  
get\_duration, YDataStream 354  
get\_dutyCycle, YPwmOutput 1090  
get\_dutyCycleAtPowerOn, YPwmOutput 1091

get\_enabled, YDisplay 423  
get\_enabled, YHubPort 666  
get\_enabled, YPwmOutput 1092  
get\_enabled, YServo 1355  
get\_enabledAtPowerOn, YPwmOutput 1093  
get\_enabledAtPowerOn, YServo 1356  
get\_endTimeUTC, YDataSet 335  
get\_endTimeUTC, YMeasure 854  
getErrorMessage, YAccelerometer 47  
getErrorMessage, YAnButton 94  
getErrorMessage, YCarbonDioxide 135  
getErrorMessage, YColorLed 174  
getErrorMessage, YCompass 211  
getErrorMessage, YCurrent 255  
getErrorMessage, YDataLogger 299  
getErrorMessage, YDigitalIO 373  
getErrorMessage, YDisplay 424  
getErrorMessage, YDualPower 498  
getErrorMessage, YFiles 530  
getErrorMessage, YGenericSensor 564  
getErrorMessage, YGyro 614  
getErrorMessage, YHubPort 667  
getErrorMessage, YHumidity 698  
getErrorMessage, YLed 738  
getErrorMessage, YLightSensor 773  
getErrorMessage, YMagnetometer 816  
getErrorMessage, YModule 870  
getErrorMessage, YNetwork 921  
getErrorMessage, YOsControl 971  
getErrorMessage, YPower 1003  
getErrorMessage, YPressure 1049  
getErrorMessage, YPwmOutput 1094  
getErrorMessage, YPwmPowerSource 1130  
getErrorMessage, YQt 1161  
getErrorMessage, YRealTimeClock 1201  
getErrorMessage, YRefFrame 1239  
getErrorMessage, YRelay 1274  
getErrorMessage, YSensor 1315  
getErrorMessage, YServo 1357  
getErrorMessage, YTemperature 1397  
getErrorMessage, YTilt 1442  
getErrorMessage, YVoc 1485  
getErrorMessage, YVoltage 1528  
getErrorMessage, YVSource 1567  
getErrorMessage, YWakeUpMonitor 1604  
getErrorMessage, YWakeUpSchedule 1643  
getErrorMessage, YWatchdog 1687  
getErrorMessage, YWireless 1735  
get\_errorType, YAccelerometer 48  
get\_errorType, YAnButton 95  
get\_errorType, YCarbonDioxide 136  
get\_errorType, YColorLed 175  
get\_errorType, YCompass 212  
get\_errorType, YCurrent 256  
get\_errorType, YDataLogger 300  
get\_errorType, YDigitalIO 374  
get\_errorType, YDisplay 425  
get\_errorType, YDualPower 499  
get\_errorType, YFiles 531  
get\_errorType, YGenericSensor 565  
get\_errorType, YGyro 615  
get\_errorType, YHubPort 668  
get\_errorType, YHumidity 699  
get\_errorType, YLed 739  
get\_errorType, YLightSensor 774  
get\_errorType, YMagnetometer 817  
get\_errorType, YModule 871  
get\_errorType, YNetwork 922  
get\_errorType, YOsControl 972  
get\_errorType, YPower 1004  
get\_errorType, YPressure 1050  
get\_errorType, YPwmOutput 1095  
get\_errorType, YPwmPowerSource 1131  
get\_errorType, YQt 1162  
get\_errorType, YRealTimeClock 1202  
get\_errorType, YRefFrame 1240  
get\_errorType, YRelay 1275  
get\_errorType, YSensor 1316  
get\_errorType, YServo 1358  
get\_errorType, YTemperature 1398  
get\_errorType, YTilt 1443  
get\_errorType, YVoc 1486  
get\_errorType, YVoltage 1529  
get\_errorType, YVSource 1568  
get\_errorType, YWakeUpMonitor 1605  
get\_errorType, YWakeUpSchedule 1644  
get\_errorType, YWatchdog 1688  
get\_errorType, YWireless 1736  
get\_extPowerFailure, YVSource 1569  
get\_extVoltage, YDualPower 500  
get\_failure, YVSource 1570  
get\_filesCount, YFiles 532  
get\_firmwareRelease, YModule 872  
get\_freeSpace, YFiles 533  
get\_frequency, YPwmOutput 1096  
get\_friendlyName, YAccelerometer 49  
get\_friendlyName, YAnButton 96  
get\_friendlyName, YCarbonDioxide 137  
get\_friendlyName, YColorLed 176  
get\_friendlyName, YCompass 213  
get\_friendlyName, YCurrent 257  
get\_friendlyName, YDataLogger 301  
get\_friendlyName, YDigitalIO 375  
get\_friendlyName, YDisplay 426  
get\_friendlyName, YDualPower 501  
get\_friendlyName, YFiles 534  
get\_friendlyName, YGenericSensor 566  
get\_friendlyName, YGyro 616  
get\_friendlyName, YHubPort 669  
get\_friendlyName, YHumidity 700  
get\_friendlyName, YLed 740  
get\_friendlyName, YLightSensor 775  
get\_friendlyName, YMagnetometer 818  
get\_friendlyName, YNetwork 923  
get\_friendlyName, YOsControl 973  
get\_friendlyName, YPower 1005  
get\_friendlyName, YPressure 1051  
get\_friendlyName, YPwmOutput 1097

get\_friendlyName, YPwmPowerSource 1132  
get\_friendlyName, YQt 1163  
get\_friendlyName, YRealTimeClock 1203  
get\_friendlyName, YRefFrame 1241  
get\_friendlyName, YRelay 1276  
get\_friendlyName, YSensor 1317  
get\_friendlyName, YServo 1359  
get\_friendlyName, YTTemperature 1399  
get\_friendlyName, YTilt 1444  
get\_friendlyName, YVoc 1487  
get\_friendlyName, YVoltage 1530  
get\_friendlyName, YVSource 1571  
get\_friendlyName, YWakeUpMonitor 1606  
get\_friendlyName, YWakeUpSchedule 1645  
get\_friendlyName, YWatchdog 1689  
get\_friendlyName, YWireless 1737  
get\_functionDescriptor, YAccelerometer 50  
get\_functionDescriptor, YAnButton 97  
get\_functionDescriptor, YCarbonDioxide 138  
get\_functionDescriptor, YColorLed 177  
get\_functionDescriptor, YCompass 214  
get\_functionDescriptor, YCurrent 258  
get\_functionDescriptor, YDataLogger 302  
get\_functionDescriptor, YDigitalIO 376  
get\_functionDescriptor, YDisplay 427  
get\_functionDescriptor, YDualPower 502  
get\_functionDescriptor, YFiles 535  
get\_functionDescriptor, YGenericSensor 567  
get\_functionDescriptor, YGyro 617  
get\_functionDescriptor, YHubPort 670  
get\_functionDescriptor, YHumidity 701  
get\_functionDescriptor, YLed 741  
get\_functionDescriptor, YLightSensor 776  
get\_functionDescriptor, YMagnetometer 819  
get\_functionDescriptor, YNetwork 924  
get\_functionDescriptor, YOsControl 974  
get\_functionDescriptor, YPower 1006  
get\_functionDescriptor, YPressure 1052  
get\_functionDescriptor, YPwmOutput 1098  
get\_functionDescriptor, YPwmPowerSource 1133  
get\_functionDescriptor, YQt 1164  
get\_functionDescriptor, YRealTimeClock 1204  
get\_functionDescriptor, YRefFrame 1242  
get\_functionDescriptor, YRelay 1277  
get\_functionDescriptor, YSensor 1318  
get\_functionDescriptor, YServo 1360  
get\_functionDescriptor, YTTemperature 1400  
get\_functionDescriptor, YTilt 1445  
get\_functionDescriptor, YVoc 1488  
get\_functionDescriptor, YVoltage 1531  
get\_functionDescriptor, YVSource 1572  
get\_functionDescriptor, YWakeUpMonitor 1607  
get\_functionDescriptor, YWakeUpSchedule 1646  
get\_functionDescriptor, YWatchdog 1690  
get\_functionDescriptor, YWireless 1738  
get\_functionId, YAccelerometer 51  
get\_functionId, YAnButton 98  
get\_functionId, YCarbonDioxide 139  
get\_functionId, YColorLed 178  
get\_functionId, YCompass 215  
get\_functionId, YCurrent 259  
get\_functionId, YDataLogger 303  
get\_functionId, YDataSet 336  
get\_functionId, YDigitalIO 377  
get\_functionId, YDisplay 428  
get\_functionId, YDualPower 503  
get\_functionId, YFiles 536  
get\_functionId, YGenericSensor 568  
get\_functionId, YGyro 618  
get\_functionId, YHubPort 671  
get\_functionId, YHumidity 702  
get\_functionId, YLed 742  
get\_functionId, YLightSensor 777  
get\_functionId, YMagnetometer 820  
get\_functionId, YNetwork 925  
get\_functionId, YOsControl 975  
get\_functionId, YPower 1007  
get\_functionId, YPressure 1053  
get\_functionId, YPwmOutput 1099  
get\_functionId, YPwmPowerSource 1134  
get\_functionId, YQt 1165  
get\_functionId, YRealTimeClock 1205  
get\_functionId, YRefFrame 1243  
get\_functionId, YRelay 1278  
get\_functionId, YSensor 1319  
get\_functionId, YServo 1361  
get\_functionId, YTTemperature 1401  
get\_functionId, YTilt 1446  
get\_functionId, YVoc 1489  
get\_functionId, YVoltage 1532  
get\_functionId, YVSource 1573  
get\_functionId, YWakeUpMonitor 1608  
get\_functionId, YWakeUpSchedule 1647  
get\_functionId, YWatchdog 1691  
get\_functionId, YWireless 1739  
get\_hardwareId, YAccelerometer 52  
get\_hardwareId, YAnButton 99  
get\_hardwareId, YCarbonDioxide 140  
get\_hardwareId, YColorLed 179  
get\_hardwareId, YCompass 216  
get\_hardwareId, YCurrent 260  
get\_hardwareId, YDataLogger 304  
get\_hardwareId, YDataSet 337  
get\_hardwareId, YDigitalIO 378  
get\_hardwareId, YDisplay 429  
get\_hardwareId, YDualPower 504  
get\_hardwareId, YFiles 537  
get\_hardwareId, YGenericSensor 569  
get\_hardwareId, YGyro 619  
get\_hardwareId, YHubPort 672  
get\_hardwareId, YHumidity 703  
get\_hardwareId, YLed 743  
get\_hardwareId, YLightSensor 778  
get\_hardwareId, YMagnetometer 821  
get\_hardwareId, YModule 873  
get\_hardwareId, YNetwork 926  
get\_hardwareId, YOsControl 976  
get\_hardwareId, YPower 1008

get\_hardwareId, YPressure 1054  
get\_hardwareId, YPwmOutput 1100  
get\_hardwareId, YPwmPowerSource 1135  
get\_hardwareId, YQt 1166  
get\_hardwareId, YRealTimeClock 1206  
get\_hardwareId, YRefFrame 1244  
get\_hardwareId, YRelay 1279  
get\_hardwareId, YSensor 1320  
get\_hardwareId, YServo 1362  
get\_hardwareId, YTemperature 1402  
get\_hardwareId, YTilt 1447  
get\_hardwareId, YVoc 1490  
get\_hardwareId, YVoltage 1533  
get\_hardwareId, YVSource 1574  
get\_hardwareId, YWakeUpMonitor 1609  
get\_hardwareId, YWakeUpSchedule 1648  
get\_hardwareId, YWatchdog 1692  
get\_hardwareId, YWireless 1740  
get\_heading, YGyro 620  
get\_highestValue, YAccelerometer 53  
get\_highestValue, YCarbonDioxide 141  
get\_highestValue, YCompass 217  
get\_highestValue, YCurrent 261  
get\_highestValue, YGenericSensor 570  
get\_highestValue, YGyro 621  
get\_highestValue, YHumidity 704  
get\_highestValue, YLightSensor 779  
get\_highestValue, YMagnetometer 822  
get\_highestValue, YPower 1009  
get\_highestValue, YPressure 1055  
get\_highestValue, YQt 1167  
get\_highestValue, YSensor 1321  
get\_highestValue, YTemperature 1403  
get\_highestValue, YTilt 1448  
get\_highestValue, YVoc 1491  
get\_highestValue, YVoltage 1534  
get\_hours, YWakeUpSchedule 1649  
get\_hslColor, YColorLed 180  
get\_icon2d, YModule 874  
get\_ipAddress, YNetwork 927  
get\_isPressed, YAnButton 100  
get\_lastLogs, YModule 875  
get\_lastTimePressed, YAnButton 101  
get\_lastTimeReleased, YAnButton 102  
get\_layerCount, YDisplay 430  
get\_layerHeight, YDisplay 431  
get\_layerHeight, YDisplayLayer 476  
get\_layerWidth, YDisplay 432  
get\_layerWidth, YDisplayLayer 477  
get\_linkQuality, YWireless 1741  
get\_list, YFiles 538  
get\_logFrequency, YAccelerometer 54  
get\_logFrequency, YCarbonDioxide 142  
get\_logFrequency, YCompass 218  
get\_logFrequency, YCurrent 262  
get\_logFrequency, YGenericSensor 571  
get\_logFrequency, YGyro 622  
get\_logFrequency, YHumidity 705  
get\_logFrequency, YLightSensor 780  
get\_logFrequency, YMagnetometer 823  
get\_logFrequency, YPower 1010  
get\_logFrequency, YPressure 1056  
get\_logFrequency, YQt 1168  
get\_logFrequency, YSensor 1322  
get\_logFrequency, YTemperature 1404  
get\_logFrequency, YTilt 1449  
get\_logFrequency, YVoc 1492  
get\_logFrequency, YVoltage 1535  
get\_logicalName, YAccelerometer 55  
get\_logicalName, YAnButton 103  
get\_logicalName, YCarbonDioxide 143  
get\_logicalName, YColorLed 181  
get\_logicalName, YCompass 219  
get\_logicalName, YCurrent 263  
get\_logicalName, YDataLogger 305  
get\_logicalName, YDigitalIO 379  
get\_logicalName, YDisplay 433  
get\_logicalName, YDualPower 505  
get\_logicalName, YFiles 539  
get\_logicalName, YGenericSensor 572  
get\_logicalName, YGyro 623  
get\_logicalName, YHubPort 673  
get\_logicalName, YHumidity 706  
get\_logicalName, YLed 744  
get\_logicalName, YLightSensor 781  
get\_logicalName, YMagnetometer 824  
get\_logicalName, YModule 876  
get\_logicalName, YNetwork 928  
get\_logicalName, YOsControl 977  
get\_logicalName, YPower 1011  
get\_logicalName, YPressure 1057  
get\_logicalName, YPwmOutput 1101  
get\_logicalName, YPwmPowerSource 1136  
get\_logicalName, YQt 1169  
get\_logicalName, YRealTimeClock 1207  
get\_logicalName, YRefFrame 1245  
get\_logicalName, YRelay 1280  
get\_logicalName, YSensor 1323  
get\_logicalName, YServo 1363  
get\_logicalName, YTemperature 1405  
get\_logicalName, YTilt 1450  
get\_logicalName, YVoc 1493  
get\_logicalName, YVoltage 1536  
get\_logicalName, YVSource 1575  
get\_logicalName, YWakeUpMonitor 1610  
get\_logicalName, YWakeUpSchedule 1650  
get\_logicalName, YWatchdog 1693  
get\_logicalName, YWireless 1742  
get\_lowestValue, YAccelerometer 56  
get\_lowestValue, YCarbonDioxide 144  
get\_lowestValue, YCompass 220  
get\_lowestValue, YCurrent 264  
get\_lowestValue, YGenericSensor 573  
get\_lowestValue, YGyro 624  
get\_lowestValue, YHumidity 707  
get\_lowestValue, YLightSensor 782  
get\_lowestValue, YMagnetometer 825  
get\_lowestValue, YPower 1012

get\_lowestValue, YPressure 1058  
get\_lowestValue, YQt 1170  
get\_lowestValue, YSensor 1324  
get\_lowestValue, YTemperature 1406  
get\_lowestValue, YTilt 1451  
get\_lowestValue, YVoc 1494  
get\_lowestValue, YVoltage 1537  
get\_luminosity, YLed 745  
get\_luminosity, YModule 877  
get\_macAddress, YNetwork 929  
get\_magneticHeading, YCompass 221  
get\_maxTimeOnStateA, YRelay 1281  
get\_maxTimeOnStateA, YWatchdog 1694  
get\_maxTimeOnStateB, YRelay 1282  
get\_maxTimeOnStateB, YWatchdog 1695  
get\_maxValue, YDataRun 326  
get\_maxValue, YDataStream 355  
get\_maxValue, YMeasure 855  
get\_measureNames, YDataRun 327  
get\_measures, YDataSet 338  
get\_message, YWireless 1743  
get\_meter, YPower 1013  
get\_meterTimer, YPower 1014  
get\_minutes, YWakeUpSchedule 1651  
get\_minutesA, YWakeUpSchedule 1652  
get\_minutesB, YWakeUpSchedule 1653  
get\_minValue, YDataRun 328  
get\_minValue, YDataStream 356  
get\_minValue, YMeasure 856  
get\_module, YAccelerometer 57  
get\_module, YAnButton 104  
get\_module, YCarbonDioxide 145  
get\_module, YColorLed 182  
get\_module, YCompass 222  
get\_module, YCurrent 265  
get\_module, YDataLogger 306  
get\_module, YDigitalIO 380  
get\_module, YDisplay 434  
get\_module, YDualPower 506  
get\_module, YFiles 540  
get\_module, YGenericSensor 574  
get\_module, YGyro 625  
get\_module, YHubPort 674  
get\_module, YHumidity 708  
get\_module, YLed 746  
get\_module, YLightSensor 783  
get\_module, YMagnetometer 826  
get\_module, YNetwork 930  
get\_module, YOsControl 978  
get\_module, YPower 1015  
get\_module, YPressure 1059  
get\_module, YPwmOutput 1102  
get\_module, YPwmPowerSource 1137  
get\_module, YQt 1171  
get\_module, YRealTimeClock 1208  
get\_module, YRefFrame 1246  
get\_module, YRelay 1283  
get\_module, YSensor 1325  
get\_module,YServo 1364  
get\_module, YTemperature 1407  
get\_module, YTilt 1452  
get\_module, YVoc 1495  
get\_module, YVoltage 1538  
get\_module, YVSource 1576  
get\_module, YWakeUpMonitor 1611  
get\_module, YWakeUpSchedule 1654  
get\_module, YWatchdog 1696  
get\_module, YWireless 1744  
get\_module\_async, YAccelerometer 58  
get\_module\_async, YAnButton 105  
get\_module\_async, YCarbonDioxide 146  
get\_module\_async, YColorLed 183  
get\_module\_async, YCompass 223  
get\_module\_async, YCurrent 266  
get\_module\_async, YDataLogger 307  
get\_module\_async, YDigitalIO 381  
get\_module\_async, YDisplay 435  
get\_module\_async, YDualPower 507  
get\_module\_async, YFiles 541  
get\_module\_async, YGenericSensor 575  
get\_module\_async, YGyro 626  
get\_module\_async, YHubPort 675  
get\_module\_async, YHumidity 709  
get\_module\_async, YLed 747  
get\_module\_async, YLightSensor 784  
get\_module\_async, YMagnetometer 827  
get\_module\_async, YNetwork 931  
get\_module\_async, YOsControl 979  
get\_module\_async, YPower 1016  
get\_module\_async, YPressure 1060  
get\_module\_async, YPwmOutput 1103  
get\_module\_async, YPwmPowerSource 1138  
get\_module\_async, YQt 1172  
get\_module\_async, YRealTimeClock 1209  
get\_module\_async, YRefFrame 1247  
get\_module\_async, YRelay 1284  
get\_module\_async, YSensor 1326  
get\_module\_async, YServo 1365  
get\_module\_async, YTemperature 1408  
get\_module\_async, YTilt 1453  
get\_module\_async, YVoc 1496  
get\_module\_async, YVoltage 1539  
get\_module\_async, YVSource 1577  
get\_module\_async, YWakeUpMonitor 1612  
get\_module\_async, YWakeUpSchedule 1655  
get\_module\_async, YWatchdog 1697  
get\_module\_async, YWireless 1745  
get\_monthDays, YWakeUpSchedule 1656  
get\_months, YWakeUpSchedule 1657  
get\_mountOrientation, YRefFrame 1248  
get\_mountPosition, YRefFrame 1249  
get\_neutral, YServo 1366  
get\_nextOccurrence, YWakeUpSchedule 1658  
get\_nextWakeUp, YWakeUpMonitor 1613  
get\_orientation, YDisplay 436  
get\_output, YRelay 1285  
get\_output, YWatchdog 1698  
get\_outputVoltage, YDigitalIO 382

get\_overCurrent, YVSource 1578  
get\_overHeat, YVSource 1579  
get\_overLoad, YVSource 1580  
get\_period, YPwmOutput 1104  
get\_persistentSettings, YModule 878  
get\_pitch, YGyro 627  
get\_poeCurrent, YNetwork 932  
get\_portDirection, YDigitalIO 383  
get\_portOpenDrain, YDigitalIO 384  
get\_portPolarity, YDigitalIO 385  
get\_portSize, YDigitalIO 386  
get\_portState, YDigitalIO 387  
get\_portState, YHubPort 676  
get\_position, YServo 1367  
get\_positionAtPowerOn, YServo 1368  
get\_power, YLed 748  
get\_powerControl, YDualPower 508  
get\_powerDuration, YWakeUpMonitor 1614  
get\_powerMode, YPwmPowerSource 1139  
get\_powerState, YDualPower 509  
get\_preview, YDataSet 339  
get\_primaryDNS, YNetwork 933  
get\_productId, YModule 879  
get\_productName, YModule 880  
get\_productRelease, YModule 881  
get\_progress, YDataSet 340  
get\_pulseCounter, YAnButton 106  
get\_pulseDuration, YPwmOutput 1105  
get\_pulseTimer, YAnButton 107  
get\_pulseTimer, YRelay 1286  
get\_pulseTimer, YWatchdog 1699  
get\_quaternionW, YGyro 628  
get\_quaternionX, YGyro 629  
get\_quaternionY, YGyro 630  
get\_quaternionZ, YGyro 631  
get\_range, YServo 1369  
get\_rawValue, YAnButton 108  
get\_readiness, YNetwork 934  
get\_rebootCountdown, YModule 882  
get\_recordedData, YAccelerometer 59  
get\_recordedData, YCarbonDioxide 147  
get\_recordedData, YCompass 224  
get\_recordedData, YCurrent 267  
get\_recordedData, YGenericSensor 576  
get\_recordedData, YGyro 632  
get\_recordedData, YHumidity 710  
get\_recordedData, YLightSensor 785  
get\_recordedData, YMagnetometer 828  
get\_recordedData, YPower 1017  
get\_recordedData, YPressure 1061  
get\_recordedData, YQt 1173  
get\_recordedData, YSensor 1327  
get\_recordedData, YTemperature 1409  
get\_recordedData, YTilt 1454  
get\_recordedData, YVoc 1497  
get\_recordedData, YVoltage 1540  
get\_recording, YDataLogger 308  
get\_regulationFailure, YVSource 1581  
get\_reportFrequency, YAccelerometer 60  
get\_reportFrequency, YCarbonDioxide 148  
get\_reportFrequency, YCompass 225  
get\_reportFrequency, YCurrent 268  
get\_reportFrequency, YGenericSensor 577  
get\_reportFrequency, YGyro 633  
get\_reportFrequency, YHumidity 711  
get\_reportFrequency, YLightSensor 786  
get\_reportFrequency, YMagnetometer 829  
get\_reportFrequency, YPower 1018  
get\_reportFrequency, YPressure 1062  
get\_reportFrequency, YQt 1174  
get\_reportFrequency, YSensor 1328  
get\_reportFrequency, YTemperature 1410  
get\_reportFrequency, YTilt 1455  
get\_reportFrequency, YVoc 1498  
get\_reportFrequency, YVoltage 1541  
get\_resolution, YAccelerometer 61  
get\_resolution, YCarbonDioxide 149  
get\_resolution, YCompass 226  
get\_resolution, YCurrent 269  
get\_resolution, YGenericSensor 578  
get\_resolution, YGyro 634  
get\_resolution, YHumidity 712  
get\_resolution, YLightSensor 787  
get\_resolution, YMagnetometer 830  
get\_resolution, YPower 1019  
get\_resolution, YPressure 1063  
get\_resolution, YQt 1175  
get\_resolution, YSensor 1329  
get\_resolution, YTemperature 1411  
get\_resolution, YTilt 1456  
get\_resolution, YVoc 1499  
get\_resolution, YVoltage 1542  
get\_rgbColor, YColorLed 184  
get\_rgbColorAtPowerOn, YColorLed 185  
get\_roll, YGyro 635  
get\_router, YNetwork 935  
getRowCount, YDataStream 357  
get\_runIndex, YDataStream 358  
get\_running, YWatchdog 1700  
get\_secondaryDNS, YNetwork 936  
get\_security, YWireless 1746  
get\_sensitivity, YAnButton 109  
get\_sensorType, YTemperature 1412  
get\_serialNumber, YModule 883  
get\_shutdownCountdown, YOsControl 980  
get\_signalRange, YGenericSensor 579  
get\_signalUnit, YGenericSensor 580  
get\_signalValue, YGenericSensor 581  
get\_sleepCountdown, YWakeUpMonitor 1615  
get\_ssid, YWireless 1747  
get\_startTime, YDataStream 359  
getStartTimeUTC, YDataRun 329  
getStartTimeUTC, YDataSet 341  
getStartTimeUTC, YDataStream 360  
getStartTimeUTC, YMeasure 857  
get\_startupSeq, YDisplay 437  
get\_state, YRelay 1287  
get\_state, YWatchdog 1701

get\_stateAtPowerOn, YRelay 1288  
get\_stateAtPowerOn, YWatchdog 1702  
get\_subnetMask, YNetwork 937  
get\_summary, YDataSet 342  
get\_timeSet, YRealTimeClock 1210  
get\_timeUTC, YDataLogger 309  
get\_triggerDelay, YWatchdog 1703  
get\_triggerDuration, YWatchdog 1704  
get\_unit, YAccelerometer 62  
get\_unit, YCarbonDioxide 150  
get\_unit, YCompass 227  
get\_unit, YCurrent 270  
get\_unit, YDataSet 343  
get\_unit, YGenericSensor 582  
get\_unit, YGyro 636  
get\_unit, YHumidity 713  
get\_unit, YLightSensor 788  
get\_unit, YMagnetometer 831  
get\_unit, YPower 1020  
get\_unit, YPressure 1064  
get\_unit, YQt 1176  
get\_unit,YSensor 1330  
get\_unit, YTemperature 1413  
get\_unit, YTilt 1457  
get\_unit, YVoc 1500  
get\_unit, YVoltage 1543  
get\_unit, YVSource 1582  
get\_unixTime, YRealTimeClock 1211  
get\_upTime, YModule 884  
get\_usbBandwidth, YModule 885  
get\_usbCurrent, YModule 886  
get\_userData, YAccelerometer 63  
get\_userData, YAnButton 110  
get\_userData, YCarbonDioxide 151  
get\_userData, YColorLed 186  
get\_userData, YCompass 228  
get\_userData, YCurrent 271  
get\_userData, YDataLogger 310  
get\_userData, YDigitalIO 388  
get\_userData, YDisplay 438  
get\_userData, YDualPower 510  
get\_userData, YFiles 542  
get\_userData, YGenericSensor 583  
get\_userData, YGyro 637  
get\_userData, YHubPort 677  
get\_userData, YHumidity 714  
get\_userData, YLed 749  
get\_userData, YLightSensor 789  
get\_userData, YMagnetometer 832  
get\_userData, YModule 887  
get\_userData, YNetwork 938  
get\_userData, YOsControl 981  
get\_userData, YPower 1021  
get\_userData, YPressure 1065  
get\_userData, YPwmOutput 1106  
get\_userData, YPwmPowerSource 1140  
get\_userData, YQt 1177  
get\_userData, YRealTimeClock 1212  
get\_userData, YRefFrame 1250

get(userData, YRelay 1289  
get(userData, YSensor 1331  
get(userData, YServo 1370  
get(userData, YTemperature 1414  
get(userData, YTilt 1458  
get(userData, YVoc 1501  
get(userData, YVoltage 1544  
get(userData, YVSource 1583  
get(userData, YWakeUpMonitor 1616  
get(userData, YWakeUpSchedule 1659  
get(userData, YWatchdog 1705  
get(userData, YWireless 1748  
get(userPassword, YNetwork 939  
get\_utcOffset, YRealTimeClock 1213  
get\_valueCount, YDataRun 330  
get\_valueInterval, YDataRun 331  
get\_valueRange, YGenericSensor 584  
get\_voltage, YVSource 1584  
get\_wakeUpReason, YWakeUpMonitor 1617  
get\_wakeUpState, YWakeUpMonitor 1618  
get\_weekDays, YWakeUpSchedule 1660  
get\_wwwWatchdogDelay, YNetwork 940  
get\_xValue, YAccelerometer 64  
get\_xValue, YGyro 638  
get\_xValue, YMagnetometer 833  
get\_yValue, YAccelerometer 65  
get\_yValue, YGyro 639  
get\_yValue, YMagnetometer 834  
get\_zValue, YAccelerometer 66  
get\_zValue, YGyro 640  
get\_zValue, YMagnetometer 835  
GetAPIVersion, YAPI 19  
GetTickCount, YAPI 20  
Gyroscope 605

## H

HandleEvents, YAPI 21  
hide, YDisplayLayer 478  
hslMove, YColorLed 187  
Humidity 689

## I

InitAPI, YAPI 22  
Integration 8  
Interface 38, 84, 126, 169, 202, 246, 289, 362, 410, 461, 493, 522, 555, 605, 660, 689, 732, 763, 807, 859, 905, 993, 1040, 1083, 1125, 1152, 1195, 1266, 1306, 1349, 1388, 1433, 1476, 1519, 1562, 1598, 1637, 1678, 1727  
Introduction 1  
isOnline, YAccelerometer 67  
isOnline, YAnButton 111  
isOnline, YCarbonDioxide 152  
isOnline, YColorLed 188  
isOnline, YCompass 229  
isOnline, YCurrent 272  
isOnline, YDataLogger 311  
isOnline, YDigitalIO 389

isOnline, YDisplay 439  
isOnline, YDualPower 511  
isOnline, YFiles 543  
isOnline, YGenericSensor 585  
isOnline, YGyro 641  
isOnline, YHubPort 678  
isOnline, YHumidity 715  
isOnline, YLed 750  
isOnline, YLightSensor 790  
isOnline, YMagnetometer 836  
isOnline, YModule 888  
isOnline, YNetwork 941  
isOnline, YOsControl 982  
isOnline, YPower 1022  
isOnline, YPressure 1066  
isOnline, YPwmOutput 1107  
isOnline, YPwmPowerSource 1141  
isOnline, YQt 1178  
isOnline, YRealTimeClock 1214  
isOnline, YRefFrame 1251  
isOnline, YRelay 1290  
isOnline, YSensor 1332  
isOnline,YServo 1371  
isOnline, YTemperature 1415  
isOnline, YTilt 1459  
isOnline, YVoc 1502  
isOnline, YVoltage 1545  
isOnline, YVSource 1585  
isOnline, YWakeUpMonitor 1619  
isOnline, YWakeUpSchedule 1661  
isOnline, YWatchdog 1706  
isOnline, YWireless 1749  
isOnline\_async, YAccelerometer 68  
isOnline\_async, YAnButton 112  
isOnline\_async, YCarbonDioxide 153  
isOnline\_async, YColorLed 189  
isOnline\_async, YCompass 230  
isOnline\_async, YCurrent 273  
isOnline\_async, YDataLogger 312  
isOnline\_async, YDigitalIO 390  
isOnline\_async, YDisplay 440  
isOnline\_async, YDualPower 512  
isOnline\_async, YFiles 544  
isOnline\_async, YGenericSensor 586  
isOnline\_async, YGyro 642  
isOnline\_async, YHubPort 679  
isOnline\_async, YHumidity 716  
isOnline\_async, YLed 751  
isOnline\_async, YLightSensor 791  
isOnline\_async, YMagnetometer 837  
isOnline\_async, YModule 889  
isOnline\_async, YNetwork 942  
isOnline\_async, YOsControl 983  
isOnline\_async, YPower 1023  
isOnline\_async, YPressure 1067  
isOnline\_async, YPwmOutput 1108  
isOnline\_async, YPwmPowerSource 1142  
isOnline\_async, YQt 1179  
isOnline\_async, YRealTimeClock 1215

isOnline\_async, YRefFrame 1252  
isOnline\_async, YRelay 1291  
isOnline\_async, YSensor 1333  
isOnline\_async, YServo 1372  
isOnline\_async, YTemperature 1416  
isOnline\_async, YTilt 1460  
isOnline\_async, YVoc 1503  
isOnline\_async, YVoltage 1546  
isOnline\_async, YVSource 1586  
isOnline\_async, YWakeUpMonitor 1620  
isOnline\_async, YWakeUpSchedule 1662  
isOnline\_async, YWatchdog 1707  
isOnline\_async, YWireless 1750

## J

joinNetwork, YWireless 1751

## L

Library 8  
LightSensor 763  
lineTo, YDisplayLayer 479  
load, YAccelerometer 69  
load, YAnButton 113  
load, YCarbonDioxide 154  
load, YColorLed 190  
load, YCompass 231  
load, YCurrent 274  
load, YDataLogger 313  
load, YDigitalIO 391  
load, YDisplay 441  
load, YDualPower 513  
load, YFiles 545  
load, YGenericSensor 587  
load, YGyro 643  
load, YHubPort 680  
load, YHumidity 717  
load, YLed 752  
load, YLightSensor 792  
load, YMagnetometer 838  
load, YModule 890  
load, YNetwork 943  
load, YOsControl 984  
load, YPower 1024  
load, YPressure 1068  
load, YPwmOutput 1109  
load, YPwmPowerSource 1143  
load, YQt 1180  
load, YRealTimeClock 1216  
load, YRefFrame 1253  
load, YRelay 1292  
load, YSensor 1334  
load, YServo 1373  
load, YTemperature 1417  
load, YTilt 1461  
load, YVoc 1504  
load, YVoltage 1547  
load, YVSource 1587  
load, YWakeUpMonitor 1621

load, YWakeUpSchedule 1663  
load, YWatchdog 1708  
load, YWireless 1752  
load\_async, YAccelerometer 71  
load\_async, YAnButton 114  
load\_async, YCarbonDioxide 156  
load\_async, YColorLed 191  
load\_async, YCompass 233  
load\_async, YCurrent 276  
load\_async, YDataLogger 314  
load\_async, YDigitalIO 392  
load\_async, YDisplay 442  
load\_async, YDualPower 514  
load\_async, YFiles 546  
load\_async, YGenericSensor 589  
load\_async, YGyro 645  
load\_async, YHubPort 681  
load\_async, YHumidity 719  
load\_async, YLed 753  
load\_async, YLightSensor 794  
load\_async, YMagnetometer 840  
load\_async, YModule 891  
load\_async, YNetwork 944  
load\_async, YOsControl 985  
load\_async, YPower 1026  
load\_async, YPressure 1070  
load\_async, YPwmOutput 1110  
load\_async, YPwmPowerSource 1144  
load\_async, YQt 1182  
load\_async, YRealTimeClock 1217  
load\_async, YRefFrame 1254  
load\_async, YRelay 1293  
load\_async,YSensor 1336  
load\_async, YServo 1374  
load\_async, YTemperature 1419  
load\_async, YTilt 1463  
load\_async, YVoc 1506  
load\_async, YVoltage 1549  
load\_async, YVSource 1588  
load\_async, YWakeUpMonitor 1622  
load\_async, YWakeUpSchedule 1664  
load\_async, YWatchdog 1709  
load\_async, YWireless 1753  
loadCalibrationPoints, YAccelerometer 70  
loadCalibrationPoints, YCarbonDioxide 155  
loadCalibrationPoints, YCompass 232  
loadCalibrationPoints, YCurrent 275  
loadCalibrationPoints, YGenericSensor 588  
loadCalibrationPoints, YGyro 644  
loadCalibrationPoints, YHumidity 718  
loadCalibrationPoints, YLightSensor 793  
loadCalibrationPoints, YMagnetometer 839  
loadCalibrationPoints, YPower 1025  
loadCalibrationPoints, YPressure 1069  
loadCalibrationPoints, YQt 1181  
loadCalibrationPoints, YSensor 1335  
loadCalibrationPoints, YTemperature 1418  
loadCalibrationPoints, YTilt 1462  
loadCalibrationPoints, YVoc 1505

loadCalibrationPoints, YVoltage 1548  
loadMore, YDataSet 344  
loadMore\_async, YDataSet 345

## M

Magnetometer 807  
Measured 853  
Module 5, 859  
more3DCalibration, YRefFrame 1255  
move, YServo 1375  
moveTo, YDisplayLayer 480

## N

Network 905  
newSequence, YDisplay 443  
nextAccelerometer, YAccelerometer 72  
nextAnButton, YAnButton 115  
nextCarbonDioxide, YCarbonDioxide 157  
nextColorLed, YColorLed 192  
nextCompass, YCompass 234  
nextCurrent, YCurrent 277  
nextDataLogger, YDataLogger 315  
nextDigitalIO, YDigitalIO 393  
nextDisplay, YDisplay 444  
nextDualPower, YDualPower 515  
nextFiles, YFiles 547  
nextGenericSensor, YGenericSensor 590  
nextGyro, YGyro 646  
nextHubPort, YHubPort 682  
nextHumidity, YHumidity 720  
nextLed, YLed 754  
nextLightSensor, YLightSensor 795  
nextMagnetometer, YMagnetometer 841  
nextModule, YModule 892  
nextNetwork, YNetwork 945  
nextOsControl, YOsControl 986  
nextPower, YPower 1027  
nextPressure, YPressure 1071  
nextPwmOutput, YPwmOutput 1111  
nextPwmPowerSource, YPwmPowerSource 1145  
nextQt, YQt 1183  
nextRealTimeClock, YRealTimeClock 1218  
nextRefFrame, YRefFrame 1256  
nextRelay, YRelay 1294  
nextSensor, YSensor 1337  
nextServo, YServo 1376  
nextTemperature, YTemperature 1420  
nextTilt, YTilt 1464  
nextVoc, YVoc 1507  
nextVoltage, YVoltage 1550  
nextVSource, YVSource 1589  
nextWakeUpMonitor, YWakeUpMonitor 1623  
nextWakeUpSchedule, YWakeUpSchedule 1665  
nextWatchdog, YWatchdog 1710  
nextWireless, YWireless 1754

## O

Object 461

## P

pauseSequence, YDisplay 445  
ping, YNetwork 946  
playSequence, YDisplay 446  
Port 660  
Power 493, 993  
PreregisterHub, YAPI 23  
Pressure 1040  
pulse, YDigitalIO 394  
pulse, YRelay 1295  
pulse, YVSource 1590  
pulse, YWatchdog 1711  
pulseDurationMove, YPwmOutput 1112  
PwmPowerSource 1125

## Q

Quaternion 1152

## R

Real 1195  
reboot, YModule 893  
Recorded 334  
Reference 12, 1226  
registerAnglesCallback, YGyro 647  
RegisterDeviceArrivalCallback, YAPI 24  
RegisterDeviceRemovalCallback, YAPI 25  
RegisterHub, YAPI 26  
RegisterHubDiscoveryCallback, YAPI 27  
registerLogCallback, YModule 894  
RegisterLogFunction, YAPI 28  
registerQuaternionCallback, YGyro 648  
registerTimedReportCallback, YAccelerometer 73  
registerTimedReportCallback, YCarbonDioxide 158  
registerTimedReportCallback, YCompass 235  
registerTimedReportCallback, YCurrent 278  
registerTimedReportCallback, YGenericSensor 591  
registerTimedReportCallback, YGyro 649  
registerTimedReportCallback, YHumidity 721  
registerTimedReportCallback, YLightSensor 796  
registerTimedReportCallback, YMagnetometer 842  
registerTimedReportCallback, YPower 1028  
registerTimedReportCallback, YPressure 1072  
registerTimedReportCallback, YQt 1184  
registerTimedReportCallback, YSensor 1338  
registerTimedReportCallback, YTemperature 1421  
registerTimedReportCallback, YTilt 1465  
registerTimedReportCallback, YVoc 1508  
registerTimedReportCallback, YVoltage 1551  
registerValueCallback, YAccelerometer 74

registerValueCallback, YAnButton 116  
registerValueCallback, YCarbonDioxide 159  
registerValueCallback, YColorLed 193  
registerValueCallback, YCompass 236  
registerValueCallback, YCurrent 279  
registerValueCallback, YDataLogger 316  
registerValueCallback, YDigitalIO 395  
registerValueCallback, YDisplay 447  
registerValueCallback, YDualPower 516  
registerValueCallback, YFiles 548  
registerValueCallback, YGenericSensor 592  
registerValueCallback, YGyro 650  
registerValueCallback, YHubPort 683  
registerValueCallback, YHumidity 722  
registerValueCallback, YLed 755  
registerValueCallback, YLightSensor 797  
registerValueCallback, YMagnetometer 843  
registerValueCallback, YNetwork 947  
registerValueCallback, YOsControl 987  
registerValueCallback, YPower 1029  
registerValueCallback, YPressure 1073  
registerValueCallback, YPwmOutput 1113  
registerValueCallback, YPwmPowerSource 1146  
registerValueCallback, YQt 1185  
registerValueCallback, YRealTimeClock 1219  
registerValueCallback, YRefFrame 1257  
registerValueCallback, YRelay 1296  
registerValueCallback, YSensor 1339  
registerValueCallback, YServo 1377  
registerValueCallback, YTemperature 1422  
registerValueCallback, YTilt 1466  
registerValueCallback, YVoc 1509  
registerValueCallback, YVoltage 1552  
registerValueCallback, YVSource 1591  
registerValueCallback, YWakeUpMonitor 1624  
registerValueCallback, YWakeUpSchedule 1666  
registerValueCallback, YWatchdog 1712  
registerValueCallback, YWireless 1755  
Relay 1266  
remove, YFiles 549  
reset, YDisplayLayer 481  
reset, YPower 1030  
resetAll, YDisplay 448  
resetCounter, YAnButton 117  
resetSleepCountDown, YWakeUpMonitor 1625  
resetWatchdog, YWatchdog 1713  
revertFromFlash, YModule 895  
rgbMove, YColorLed 194

## S

save3DCalibration, YRefFrame 1258  
saveSequence, YDisplay 449  
saveToFlash, YModule 896  
SelectArchitecture, YAPI 29  
selectColorPen, YDisplayLayer 482  
selectEraser, YDisplayLayer 483  
selectFont, YDisplayLayer 484  
selectGrayPen, YDisplayLayer 485  
Sensor 1306

Sequence 324, 334, 347  
Servo 1349  
set\_adminPassword, YNetwork 948  
set\_analogCalibration, YAnButton 118  
set\_autoStart, YDataLogger 317  
set\_autoStart, YWatchdog 1714  
set\_beacon, YModule 897  
set\_bearing, YRefFrame 1259  
set\_bitDirection, YDigitalIO 396  
set\_bitOpenDrain, YDigitalIO 397  
set\_bitPolarity, YDigitalIO 398  
set\_bitState, YDigitalIO 399  
set\_blinking, YLed 756  
set\_brightness, YDisplay 450  
set\_calibrationMax, YAnButton 119  
set\_calibrationMin, YAnButton 120  
set\_callbackCredentials, YNetwork 949  
set\_callbackEncoding, YNetwork 950  
set\_callbackMaxDelay, YNetwork 951  
set\_callbackMethod, YNetwork 952  
set\_callbackMinDelay, YNetwork 953  
set\_callbackUrl, YNetwork 954  
set\_discoverable, YNetwork 955  
set\_dutyCycle, YPwmOutput 1114  
set\_dutyCycleAtPowerOn, YPwmOutput 1115  
set\_enabled, YDisplay 451  
set\_enabled, YHubPort 684  
set\_enabled, YPwmOutput 1116  
set\_enabled,YServo 1378  
set\_enabledAtPowerOn, YPwmOutput 1117  
set\_enabledAtPowerOn,YServo 1379  
set\_frequency, YPwmOutput 1118  
set\_highestValue, YAccelerometer 75  
set\_highestValue, YCarbonDioxide 160  
set\_highestValue, YCompass 237  
set\_highestValue, YCurrent 280  
set\_highestValue, YGenericSensor 593  
set\_highestValue, YGyro 651  
set\_highestValue, YHumidity 723  
set\_highestValue, YLightSensor 798  
set\_highestValue, YMagnetometer 844  
set\_highestValue, YPower 1031  
set\_highestValue, YPressure 1074  
set\_highestValue, YQt 1186  
set\_highestValue, YSensor 1340  
set\_highestValue, YTTemperature 1423  
set\_highestValue, YTilt 1467  
set\_highestValue, YVoc 1510  
set\_highestValue, YVoltage 1553  
set\_hours, YWakeUpSchedule 1667  
set\_hslColor, YColorLed 195  
set\_logFrequency, YAccelerometer 76  
set\_logFrequency, YCarbonDioxide 161  
set\_logFrequency, YCompass 238  
set\_logFrequency, YCurrent 281  
set\_logFrequency, YGenericSensor 594  
set\_logFrequency, YGyro 652  
set\_logFrequency, YHumidity 724  
set\_logFrequency, YLightSensor 799  
set\_logFrequency, YMagnetometer 845  
set\_logFrequency, YPower 1032  
set\_logFrequency, YPressure 1075  
set\_logFrequency, YQt 1187  
set\_logFrequency, YSensor 1341  
set\_logFrequency, YTTemperature 1424  
set\_logFrequency, YTilt 1468  
set\_logFrequency, YVoc 1511  
set\_logFrequency, YVoltage 1554  
set\_logicalName, YAccelerometer 77  
set\_logicalName, YAnButton 121  
set\_logicalName, YCarbonDioxide 162  
set\_logicalName, YColorLed 196  
set\_logicalName, YCompass 239  
set\_logicalName, YCurrent 282  
set\_logicalName, YDataLogger 318  
set\_logicalName, YDigitalIO 400  
set\_logicalName, YDisplay 452  
set\_logicalName, YDualPower 517  
set\_logicalName, YFiles 550  
set\_logicalName, YGenericSensor 595  
set\_logicalName, YGyro 653  
set\_logicalName, YHubPort 685  
set\_logicalName, YHumidity 725  
set\_logicalName, YLed 757  
set\_logicalName, YLightSensor 800  
set\_logicalName, YMagnetometer 846  
set\_logicalName, YModule 898  
set\_logicalName, YNetwork 956  
set\_logicalName, YOsControl 988  
set\_logicalName, YPower 1033  
set\_logicalName, YPressure 1076  
set\_logicalName, YPwmOutput 1119  
set\_logicalName, YPwmPowerSource 1147  
set\_logicalName, YQt 1188  
set\_logicalName, YRealTimeClock 1220  
set\_logicalName, YRefFrame 1260  
set\_logicalName, YRelay 1297  
set\_logicalName, YSensor 1342  
set\_logicalName, YServo 1380  
set\_logicalName, YTTemperature 1425  
set\_logicalName, YTilt 1469  
set\_logicalName, YVoc 1512  
set\_logicalName, YVoltage 1555  
set\_logicalName, YVSource 1592  
set\_logicalName, YWakeUpMonitor 1626  
set\_logicalName, YWakeUpSchedule 1668  
set\_logicalName, YWatchdog 1715  
set\_logicalName, YWireless 1756  
set\_lowestValue, YAccelerometer 78  
set\_lowestValue, YCarbonDioxide 163  
set\_lowestValue, YCompass 240  
set\_lowestValue, YCurrent 283  
set\_lowestValue, YGenericSensor 596  
set\_lowestValue, YGyro 654  
set\_lowestValue, YHumidity 726  
set\_lowestValue, YLightSensor 801  
set\_lowestValue, YMagnetometer 847  
set\_lowestValue, YPower 1034

set\_lowestValue, YPressure 1077  
set\_lowestValue, YQt 1189  
set\_lowestValue, YSensor 1343  
set\_lowestValue, YTemperature 1426  
set\_lowestValue, YTilt 1470  
set\_lowestValue, YVoc 1513  
set\_lowestValue, YVoltage 1556  
set\_luminosity, YLed 758  
set\_luminosity, YModule 899  
set\_maxTimeOnStateA, YRelay 1298  
set\_maxTimeOnStateA, YWatchdog 1716  
set\_maxTimeOnStateB, YRelay 1299  
set\_maxTimeOnStateB, YWatchdog 1717  
set\_minutes, YWakeUpSchedule 1669  
set\_minutesA, YWakeUpSchedule 1670  
set\_minutesB, YWakeUpSchedule 1671  
set\_monthDays, YWakeUpSchedule 1672  
set\_months, YWakeUpSchedule 1673  
set\_mountPosition, YRefFrame 1261  
set\_neutral,YServo 1381  
set\_nextWakeUp, YWakeUpMonitor 1627  
set\_orientation, YDisplay 453  
set\_output, YRelay 1300  
set\_output, YWatchdog 1718  
set\_outputVoltage, YDigitalIO 401  
set\_period, YPwmOutput 1120  
set\_portDirection, YDigitalIO 402  
set\_portOpenDrain, YDigitalIO 403  
set\_portPolarity, YDigitalIO 404  
set\_portState, YDigitalIO 405  
set\_position, YServo 1382  
set\_positionAtPowerOn, YServo 1383  
set\_power, YLed 759  
set\_powerControl, YDualPower 518  
set\_powerDuration, YWakeUpMonitor 1628  
set\_powerMode, YPwmPowerSource 1148  
set\_primaryDNS, YNetwork 957  
set\_pulseDuration, YPwmOutput 1121  
set\_range, YServo 1384  
set\_recording, YDataLogger 319  
set\_reportFrequency, YAccelerometer 79  
set\_reportFrequency, YCarbonDioxide 164  
set\_reportFrequency, YCompass 241  
set\_reportFrequency, YCurrent 284  
set\_reportFrequency, YGenericSensor 597  
set\_reportFrequency, YGyro 655  
set\_reportFrequency, YHumidity 727  
set\_reportFrequency, YLightSensor 802  
set\_reportFrequency, YMagnetometer 848  
set\_reportFrequency, YPower 1035  
set\_reportFrequency, YPressure 1078  
set\_reportFrequency, YQt 1190  
set\_reportFrequency, YSensor 1344  
set\_reportFrequency, YTemperature 1427  
set\_reportFrequency, YTilt 1471  
set\_reportFrequency, YVoc 1514  
set\_reportFrequency, YVoltage 1557  
set\_resolution, YAccelerometer 80  
set\_resolution, YCarbonDioxide 165  
set\_resolution, YCompass 242  
set\_resolution, YCurrent 285  
set\_resolution, YGenericSensor 598  
set\_resolution, YGyro 656  
set\_resolution, YHumidity 728  
set\_resolution, YLightSensor 803  
set\_resolution, YMagnetometer 849  
set\_resolution, YPower 1036  
set\_resolution, YPressure 1079  
set\_resolution, YQt 1191  
set\_resolution, YSensor 1345  
set\_resolution, YTemperature 1428  
set\_resolution, YTilt 1472  
set\_resolution, YVoc 1515  
set\_resolution, YVoltage 1558  
set\_rgbColor, YColorLed 197  
set\_rgbColorAtPowerOn, YColorLed 198  
set\_running, YWatchdog 1719  
set\_secondaryDNS, YNetwork 958  
set\_sensitivity, YAnButton 122  
set\_sensorType, YTemperature 1429  
set\_signalRange, YGenericSensor 599  
set\_sleepCountdown, YWakeUpMonitor 1629  
set\_startupSeq, YDisplay 454  
set\_state, YRelay 1301  
set\_state, YWatchdog 1720  
set\_stateAtPowerOn, YRelay 1302  
set\_stateAtPowerOn, YWatchdog 1721  
set\_timeUTC, YDataLogger 320  
set\_triggerDelay, YWatchdog 1722  
set\_triggerDuration, YWatchdog 1723  
set\_unit, YGenericSensor 600  
set\_unixTime, YRealTimeClock 1221  
set\_usbBandwidth, YModule 900  
set\_userData, YAccelerometer 81  
set\_userData, YAnButton 123  
set\_userData, YCarbonDioxide 166  
set\_userData, YColorLed 199  
set\_userData, YCompass 243  
set\_userData, YCurrent 286  
set\_userData, YDataLogger 321  
set\_userData, YDigitalIO 406  
set\_userData, YDisplay 455  
set\_userData, YDualPower 519  
set\_userData, YFiles 551  
set\_userData, YGenericSensor 601  
set\_userData, YGyro 657  
set\_userData, YHubPort 686  
set\_userData, YHumidity 729  
set\_userData, YLed 760  
set\_userData, YLightSensor 804  
set\_userData, YMagnetometer 850  
set\_userData, YModule 901  
set\_userData, YNetwork 959  
set\_userData, YOsControl 989  
set\_userData, YPower 1037  
set\_userData, YPressure 1080  
set\_userData, YPwmOutput 1122  
set\_userData, YPwmPowerSource 1149

set(userData, YQt 1192  
set(userData, YRealTimeClock 1222  
set(userData, YRefFrame 1262  
set(userData, YRelay 1303  
set(userData, YSensor 1346  
set(userData,YServo 1385  
set(userData, YTemperature 1430  
set(userData, YTilt 1473  
set(userData, YVoc 1516  
set(userData, YVoltage 1559  
set(userData, YVSource 1593  
set(userData, YWakeUpMonitor 1630  
set(userData, YWakeUpSchedule 1674  
set(userData, YWatchdog 1724  
set(userData, YWireless 1757  
set(userPassword, YNetwork 960  
set(utcOffset, YRealTimeClock 1223  
set(valueInterval, YDataRun 332  
set(valueRange, YGenericSensor 602  
set(voltage, YVSource 1594  
set(weekDays, YWakeUpSchedule 1675  
set(wwwWatchdogDelay, YNetwork 961  
setAntialiasingMode, YDisplayLayer 486  
setConsoleBackground, YDisplayLayer 487  
setConsoleMargins, YDisplayLayer 488  
setConsoleWordWrap, YDisplayLayer 489  
SetDelegate, YAPI 30  
setLayerPosition, YDisplayLayer 490  
SetTimeout, YAPI 31  
shutdown, YOsControl 990  
Sleep, YAPI 32  
sleep, YWakeUpMonitor 1631  
sleepFor, YWakeUpMonitor 1632  
sleepUntil, YWakeUpMonitor 1633  
Source 1562  
start3DCalibration, YRefFrame 1263  
stopSequence, YDisplay 456  
Supply 493  
swapLayerContent, YDisplay 457

## T

Temperature 1388  
Tilt 1433  
Time 1195  
toggle\_bitState, YDigitalIO 407  
triggerFirmwareUpdate, YModule 902  
TriggerHubDiscovery, YAPI 33

## U

Unformatted 347  
unhide, YDisplayLayer 491  
UnregisterHub, YAPI 34  
UpdateDeviceList, YAPI 35  
UpdateDeviceList\_async, YAPI 36  
upload, YDisplay 458  
upload, YFiles 552  
useDHCP, YNetwork 962  
useStaticIP, YNetwork 963

## V

Value 853  
Variants 8  
Voltage 1519, 1562  
voltageMove, YVSource 1595

## W

wait\_async, YAccelerometer 82  
wait\_async, YAnButton 124  
wait\_async, YCarbonDioxide 167  
wait\_async, YColorLed 200  
wait\_async, YCompass 244  
wait\_async, YCurrent 287  
wait\_async, YDataLogger 322  
wait\_async, YDigitalIO 408  
wait\_async, YDisplay 459  
wait\_async, YDualPower 520  
wait\_async, YFiles 553  
wait\_async, YGenericSensor 603  
wait\_async, YGyro 658  
wait\_async, YHubPort 687  
wait\_async, YHumidity 730  
wait\_async, YLed 761  
wait\_async, YLightSensor 805  
wait\_async, YMagnetometer 851  
wait\_async, YModule 903  
wait\_async, YNetwork 964  
wait\_async, YOsControl 991  
wait\_async, YPower 1038  
wait\_async, YPressure 1081  
wait\_async, YPwmOutput 1123  
wait\_async, YPwmPowerSource 1150  
wait\_async, YQt 1193  
wait\_async, YRealTimeClock 1224  
wait\_async, YRefFrame 1264  
wait\_async, YRelay 1304  
wait\_async, YSensor 1347  
wait\_async, YServo 1386  
wait\_async, YTemperature 1431  
wait\_async, YTilt 1474  
wait\_async, YVoc 1517  
wait\_async, YVoltage 1560  
wait\_async, YVSource 1596  
wait\_async, YWakeUpMonitor 1634  
wait\_async, YWakeUpSchedule 1676  
wait\_async, YWatchdog 1725  
wait\_async, YWireless 1758  
wakeUp, YWakeUpMonitor 1635  
WakeUpMonitor 1598  
WakeUpSchedule 1637  
Watchdog 1678  
Wireless 1727

## Y

YAccelerometer 40-82  
YAnButton 86-124

YAPI 14-36  
YCarbonDioxide 128-167  
yCheckLogicalName 14  
YColorLed 170-200  
YCompass 204-244  
YCurrent 248-287  
YDataLogger 290-322  
YDataRun 324-332  
YDataSet 335-345  
YDataStream 348-360  
YDigitalIO 364-408  
yDisableExceptions 15  
YDisplay 412-459  
YDisplayLayer 462-491  
YDualPower 494-520  
yEnableExceptions 16  
yEnableUSBHost 17  
YFiles 523-553  
yFindAccelerometer 40  
yFindAnButton 86  
yFindCarbonDioxide 128  
yFindColorLed 170  
yFindCompass 204  
yFindCurrent 248  
yFindDataLogger 290  
yFindDigitalIO 364  
yFindDisplay 412  
yFindDualPower 494  
yFindFiles 523  
yFindGenericSensor 557  
yFindGyro 607  
yFindHubPort 661  
yFindHumidity 691  
yFindLed 733  
yFindLightSensor 765  
yFindMagnetometer 809  
yFindModule 861  
yFindNetwork 908  
yFindOsControl 967  
yFindPower 995  
yFindPressure 1042  
yFindPwmOutput 1085  
yFindPwmPowerSource 1126  
yFindQt 1154  
yFindRealTimeClock 1196  
yFindRefFrame 1228  
yFindRelay 1268  
yFindSensor 1308  
yFindServo 1351  
yFindTemperature 1390  
yFindTilt 1435  
yFindVoc 1478  
yFindVoltage 1521  
yFindVSource 1563  
yFindWakeUpMonitor 1600  
yFindWakeUpSchedule 1639  
yFindWatchdog 1680  
yFindWireless 1728  
yFirstAccelerometer 41  
yFirstAnButton 87  
yFirstCarbonDioxide 129  
yFirstColorLed 171  
yFirstCompass 205  
yFirstCurrent 249  
yFirstDataLogger 291  
yFirstDigitalIO 365  
yFirstDisplay 413  
yFirstDualPower 495  
yFirstFiles 524  
yFirstGenericSensor 558  
yFirstGyro 608  
yFirstHubPort 662  
yFirstHumidity 692  
yFirstLed 734  
yFirstLightSensor 766  
yFirstMagnetometer 810  
yFirstModule 862  
yFirstNetwork 909  
yFirstOsControl 968  
yFirstPower 996  
yFirstPressure 1043  
yFirstPwmOutput 1086  
yFirstPwmPowerSource 1127  
yFirstQt 1155  
yFirstRealTimeClock 1197  
yFirstRefFrame 1229  
yFirstRelay 1269  
yFirstSensor 1309  
yFirstServo 1352  
yFirstTemperature 1391  
yFirstTilt 1436  
yFirstVoc 1479  
yFirstVoltage 1522  
yFirstVSource 1564  
yFirstWakeUpMonitor 1601  
yFirstWakeUpSchedule 1640  
yFirstWatchdog 1681  
yFirstWireless 1729  
yFreeAPI 18  
YGenericSensor 557-603  
yGetAPIVersion 19  
yGetTickCount 20  
YGyro 607-658  
yHandleEvents 21  
YHubPort 661-687  
YHumidity 691-730  
yInitAPI 22  
YLed 733-761  
YLightSensor 765-805  
YMagnetometer 809-851  
YMeasure 853-857  
YModule 861-903  
YNetwork 908-964  
Yocto-Demo 3  
Yocto-hub 660  
YOsControl 967-991  
YPower 995-1038  
yPreregisterHub 23

YPressure 1042-1081  
YPwmOutput 1085-1123  
YPwmPowerSource 1126-1150  
YQt 1154-1193  
YRealTimeClock 1196-1224  
YRefFrame 1228-1264  
yRegisterDeviceArrivalCallback 24  
yRegisterDeviceRemovalCallback 25  
yRegisterHub 26  
yRegisterHubDiscoveryCallback 27  
yRegisterLogFunction 28  
YRelay 1268-1304  
ySelectArchitecture 29  
YSensor 1308-1347  
YServo 1351-1386

ySetDelegate 30  
ySetTimeout 31  
ySleep 32  
YTemperature 1390-1431  
YTilt 1435-1474  
yTriggerHubDiscovery 33  
yUnregisterHub 34  
yUpdateDeviceList 35  
yUpdateDeviceList\_async 36  
YVoc 1478-1517  
YVoltage 1521-1560  
YVSource 1563-1596  
YWakeUpMonitor 1600-1635  
YWakeUpSchedule 1639-1676  
YWatchdog 1680-1725  
YWireless 1728-1758