



Référence de l'API PYTHON



# Table des matières

<b>1. Introduction</b>	<b>1</b>
<b>2. Utilisation du Yocto-Demo en Python</b>	<b>3</b>
2.1. Fichiers sources	3
2.2. Librairie dynamique	3
2.3. Contrôle de la fonction Led	4
2.4. Contrôle de la partie module	5
2.5. Gestion des erreurs	7
Blueprint	10
<b>3. Reference</b>	<b>10</b>
3.1. Fonctions générales	11
3.2. Interface de la fonction Accelerometer	32
3.3. Interface de la fonction AnButton	74
3.4. Interface de la fonction CarbonDioxide	112
3.5. Interface de la fonction ColorLed	151
3.6. Interface de la fonction Compass	180
3.7. Interface de la fonction Current	220
3.8. Interface de la fonction DataLogger	259
3.9. Séquence de données mise en forme	290
3.10. Séquence de données enregistrées	300
3.11. Séquence de données enregistrées brute	312
3.12. Interface de la fonction DigitalIO	327
3.13. Interface de la fonction Display	371
3.14. Interface des objets DisplayLayer	418
3.15. Interface de contrôle de l'alimentation	450
3.16. Interface de la fonction Files	475
3.17. Interface de la fonction GenericSensor	503
3.18. Interface de la fonction Gyro	549
3.19. Interface d'un port de Yocto-hub	600
3.20. Interface de la fonction Humidity	625
3.21. Interface de la fonction Led	664
3.22. Interface de la fonction LightSensor	691
3.23. Interface de la fonction Magnetometer	731
3.24. Valeur mesurée	773

3.25. Interface de contrôle du module .....	779
3.26. Interface de la fonction Network .....	822
3.27. contrôle d'OS .....	879
3.28. Interface de la fonction Power .....	902
3.29. Interface de la fonction Pressure .....	945
3.30. Interface de la fonction Pwm .....	984
3.31. Interface de la fonction PwmPowerSource .....	1022
3.32. Interface du quaternion .....	1045
3.33. Interface de la fonction Horloge Temps Real .....	1084
3.34. Configuration du référentiel .....	1111
3.35. Interface de la fonction Relay .....	1147
3.36. Interface des fonctions de type senseur .....	1183
3.37. Interface de la fonction Servo .....	1222
3.38. Interface de la fonction Temperature .....	1257
3.39. Interface de la fonction Tilt .....	1298
3.40. Interface de la fonction Voc .....	1337
3.41. Interface de la fonction Voltage .....	1376
3.42. Interface de la fonction Source de tension .....	1415
3.43. Interface de la fonction WakeUpMonitor .....	1443
3.44. Interface de la fonction WakeUpSchedule .....	1478
3.45. Interface de la fonction Watchdog .....	1515
3.46. Interface de la fonction Wireless .....	1560

<b>Index .....</b>	<b>1589</b>
--------------------	-------------

# 1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie Python de Yoctopuce pour interfacer vos senseurs et contrôleurs USB.

Le chapitre suivant reprend un chapitre du manuel du module USB gratuit Yocto-Demo, afin d'illustrer l'utilisation de la librairie sur des exemples concrets.

Le reste du manuel documente chaque fonction, classe et méthode de l'API. La première section décrit les fonctions globales d'ordre général, et les sections décrivent les différentes classes, utiles selon le module Yoctopuce utilisé. Pour plus d'informations sur la signification et l'utilisation d'un attribut particulier d'un module, il est recommandé de se référer à la documentation spécifique du module, qui contient plus de détails.



## 2. Utilisation du Yocto-Demo en Python

Python est un langage interprété orienté objet développé par Guido van Rossum. Il offre l'avantage d'être gratuit et d'être disponible pour la plupart de plate-formes tant Windows qu'Unix. C'est un langage idéal pour écrire des petits scripts sur un coin de table. La librairie Yoctopuce est compatible avec Python 2.6+ et 3+. Elle fonctionne sous Windows, Mac OS X et Linux tant Intel qu'ARM. La librairie a été testée avec Python 2.6 et Python 3.2. Les interpréteurs Python sont disponibles sur le site de Python <sup>1</sup>.

### 2.1. Fichiers sources

Les classes de la librairie Yoctopuce<sup>2</sup> pour Python que vous utiliserez vous sont fournies au format source. Copiez tout le contenu du répertoire *Sources* dans le répertoire de votre choix et ajoutez ce répertoire à la variable d'environnement *PYTHONPATH*. Si vous utilisez un IDE pour programmer en Python, référez-vous à sa documentation afin de le configurer de manière à ce qu'il retrouve automatiquement les fichiers sources de l'API.

### 2.2. Librairie dynamique

Une partie de la librairie de bas-niveau est écrite en C, mais vous n'aurez a priori pas besoin d'interagir directement avec elle: cette partie est fournie sous forme de DLL sous Windows, de fichier *.so* sous Unix et de fichier *.dylib* sous Mac OS X. Tout a été fait pour que l'interaction avec cette librairie se fasse aussi simplement que possible depuis Python: les différentes versions de la librairie dynamique correspondant aux différents systèmes d'exploitation et architectures sont stockées dans le répertoire *cdll*. L'API va charger automatiquement le bon fichier lors de son initialisation. Vous n'aurez donc pas à vous en soucier.

Si un jour vous deviez vouloir recompiler la librairie dynamique, vous trouverez tout son code source dans la librairie Yoctopuce pour le C++.

Afin de les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que le fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique.

---

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

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

## 2.3. Contrôle de la fonction Led

Il suffit de quelques lignes de code pour piloter un Yocto-Demo. Voici le squelette d'un fragment de code Python qui utilise la fonction Led.

```
[...]

errmsg=YRefParam()
#On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb",errmsg)
led = YLed.FindLed("YCTOPOC1-123456.led")

#Pour gérer le hot-plug, on vérifie que le module est là
if led.isOnline():
    #Use led.set_power()
    ...

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

### YAPI.RegisterHub

La fonction `YAPI.RegisterHub` initialise l'API de Yoctopuce en indiquant où les modules doivent être recherchés. Utilisée avec le paramètre "usb", elle permet de travailler avec les modules connectés localement à la machine. Si l'initialisation se passe mal, cette fonction renverra une valeur différente de `YAPI.SUCCESS`, et retournera via l'objet `errmsg` une explication du problème.

### YLed.FindLed

La fonction `YLed.FindLed`, permet de retrouver une led en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = YLed.FindLed("YCTOPOC1-123456.led")
led = YLed.FindLed("YCTOPOC1-123456.MaFonction")
led = YLed.FindLed("MonModule.led")
led = YLed.FindLed("MonModule.MaFonction")
led = YLed.FindLed("MaFonction")
```

`YLed.FindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

### isOnline

La méthode `YLed.isOnline()` de l'objet renvoyé par `FindLed` permet de savoir si le module correspondant est présent et en état de marche.

### set\_power

La fonction `set_power()` de l'objet renvoyé par `YLed.FindLed` permet d'allumer et d'éteindre la led. L'argument est `YLed.POWER_ON` ou `YLed.POWER_OFF`. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

### Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce.

Vous reconnaîtrez dans cet exemple l'utilisation des fonctions expliquées ci-dessus, cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.



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

def usage():
    scriptname = os.path.basename(sys.argv[0])
    print("Usage:")
    print(scriptname+' <serial_number>')
    print(scriptname+' <logical_name>')
    print(scriptname+' any ')
    sys.exit()

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

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

errmsg=YRefParam()

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

target=sys.argv[1]

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

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

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

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

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

c= input("command:")

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

## 2.4. Contrôle de la partie module

Chaque module peut-être contrôlé d'une manière similaire, vous trouverez ci-dessous un simple programme d'exemple affichant les principaux paramètres d'un module et permettant d'activer la balise de localisation.

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

def usage():
```

```

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

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

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

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

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

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

```

Chaque propriété xxx du module peut être lue grâce à une méthode du type `YModule.get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `YModule.set_xxx()`. Pour plus de détails concernant ces fonctions utilisées, reportez-vous au chapitre API.

## Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `YModule.set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `YModule.saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `YModule.revertFromFlash()`. Ce petit exemple ci-dessous vous permet de changer le nom logique d'un module.

```

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

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

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

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

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

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

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

## Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `YModule.yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la méthode `nextModule()` de cet objet pour trouver les modules suivants, et ce tant que la réponse n'est pas un `null`. Ci-dessous un petit exemple listant les modules connectés

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

from yocto_api import *

errmsg=YRefParam()

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

print('Device list')

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

## 2.5. Gestion des erreurs

Lorsque vous implémentez un programme qui doit interagir avec des modules USB, vous ne pouvez pas faire abstraction de la gestion des erreurs. Il y aura forcément une occasion où un utilisateur aura débranché le périphérique, soit avant de lancer le programme, soit même en pleine opération. La librairie Yoctopuce est prévue pour vous aider à supporter ce genre de comportements, mais votre code doit néanmoins être fait pour se comporter au mieux pour interpréter les erreurs signalées par la librairie.

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme. La seule manière de l'éviter est d'implémenter une des deux techniques de gestion des erreurs décrites ci-dessous.

La méthode recommandée par la plupart des langages de programmation pour la gestion des erreurs imprévisibles est l'utilisation d'exceptions. C'est le comportement par défaut de la librairie Yoctopuce. Si une erreur se produit alors qu'on essaie d'accéder à un module, la librairie va lancer une exception. Dans ce cas, de trois choses l'une:

- Si votre code attrape l'exception au vol et la gère, et tout se passe bien.
- Si votre programme tourne dans le debugger, vous pourrez relativement facilement déterminer où le problème s'est produit, et voir le message explicatif lié à l'exception.
- Sinon... l'exception va crasher votre programme, boum!

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `yDisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des méthodes `errType()` et `errMessage()`. Ce sont les mêmes informations qui auraient été associées à l'exception si elles avaient été actives.



### **3. Reference**

## 3.1. Fonctions générales

Ces quelques fonctions générales permettent l'initialisation et la configuration de la librairie Yoctopuce. Dans la plupart des cas, un appel à `yRegisterHub()` suffira en tout et pour tout. Ensuite, vous pourrez appeler la fonction globale `yFind...()` ou `yFirst...()` correspondant à votre module pour pouvoir interagir avec lui.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

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

Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.

#### **yDisableExceptions()**

Désactive l'utilisation d'exceptions pour la gestion des erreurs.

#### **yEnableExceptions()**

Réactive l'utilisation d'exceptions pour la gestion des erreurs.

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

Cette fonction est utilisée uniquement sous Android.

#### **yFreeAPI()**

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

#### **yGetAPIVersion()**

Retourne la version de la librairie Yoctopuce utilisée.

#### **yGetTickCount()**

Retourne la valeur du compteur monotone de temps (en millisecondes).

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

Maintient la communication de la librairie avec les modules Yoctopuce.

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

Initialise la librairie de programmation de Yoctopuce explicitement.

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

Alternative plus tolérante à `RegisterHub()`.

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

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est branché.

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

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est débranché.

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

Configure la librairie Yoctopuce pour utiliser les modules connectés sur une machine donnée.

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

### 3. Reference

Enregistre une fonction de callback qui est appelée chaque fois qu'un hub réseau s'annonce avec un message SSDP.

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

Enregistre une fonction de callback qui sera appelée à chaque fois que l'API a quelque chose à dire.

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

Sélectionne manuellement l'architecture de la librairie dynamique à utiliser pour accéder à USB.

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

(Objective-C uniquement) Enregistre un objet délégué qui doit se conformer au protocole YDeviceHotPlug.

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

Appelle le callback spécifié après un temps d'attente spécifié.

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

Effectue une pause dans l'exécution du programme pour une durée spécifiée.

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

Relance une détection des hubs réseau.

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

Configure la librairie Yoctopuce pour ne plus utiliser les modules connectés sur une machine préalablement enregistrer avec RegisterHub.

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

Force une mise-à-jour de la liste des modules Yoctopuce connectés.

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

Force une mise-à-jour de la liste des modules Yoctopuce connectés.



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

Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.

```
def CheckLogicalName( name)
```

Un nom logique valide est formé de 19 caractères au maximum, choisis parmi `A . . Z`, `a . . z`, `0 . . 9`, `_` et `-`. Lorsqu'on configure un nom logique avec une chaîne incorrecte, les caractères invalides sont ignorés.

**Paramètres :**

**name** une chaîne de caractères contenant le nom vérifier.

**Retourne :**

`true` si le nom est valide, `false` dans le cas contraire.

## YAPI.DisableExceptions()

YAPI

### yDisableExceptions()YAPI.DisableExceptions()

---

Désactive l'utilisation d'exceptions pour la gestion des erreurs.

```
def DisableExceptions( )
```

Lorsque les exceptions sont désactivées, chaque fonction retourne une valeur d'erreur spécifique selon son type, documentée dans ce manuel de référence.

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

---

Réactive l'utilisation d'exceptions pour la gestion des erreurs.

```
def EnableExceptions( )
```

Attention, lorsque les exceptions sont activées, tout appel à une fonction de la librairie qui échoue déclenche une exception. Dans le cas où celle-ci n'est pas interceptée correctement par le code appelant, soit le debugger se lance, soit le programme de l'utilisateur est immédiatement stoppé (crash).

**YAPI.FreeAPI()**

**YAPI**

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

---

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

def **FreeAPI**( )

Il n'est en général pas nécessaire d'appeler cette fonction, sauf si vous désirez libérer tous les blocs de mémoire alloués dynamiquement dans le but d'identifier une source de blocs perdus par exemple. Vous ne devez plus appeler aucune fonction de la librairie après avoir appelé `yFreeAPI( )`, sous peine de crash.

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

Retourne la version de la librairie Yoctopuce utilisée.

```
def GetAPIVersion( )
```

La version est retournée sous forme d'une chaîne de caractères au format "Majeure.Mineure.NoBuild", par exemple "1.01.5535". Pour les langages utilisant une DLL externe (par exemple C#, VisualBasic ou Delphi), la chaîne contient en outre la version de la DLL au même format, par exemple "1.01.5535 (1.01.5439)".

Si vous désirez vérifier dans votre code que la version de la librairie est compatible avec celle que vous avez utilisé durant le développement, vérifiez que le numéro majeur soit strictement égal et que le numéro mineur soit égal ou supérieur. Le numéro de build n'est pas significatif par rapport à la compatibilité de la librairie.

**Retourne :**

une chaîne de caractères décrivant la version de la librairie.

## YAPI.GetTickCount()

YAPI

### yGetTickCount()YAPI.GetTickCount()

---

Retourne la valeur du compteur monotone de temps (en millisecondes).

```
def GetTickCount( )
```

Ce compteur peut être utilisé pour calculer des délais en rapport avec les modules Yoctopuce, dont la base de temps est aussi la milliseconde.

**Retourne :**

un long entier contenant la valeur du compteur de millisecondes.

## YAPI.HandleEvents() yHandleEvents()YAPI.HandleEvents()

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

```
def HandleEvents( errmsg=None)
```

Si votre programme inclut des longues boucles d'attente, vous pouvez y inclure un appel à cette fonction pour que la librairie prenne en charge les informations mise en attente par les modules sur les canaux de communication. Ce n'est pas strictement indispensable mais cela peut améliorer la réactivité des la librairie pour les commandes suivantes.

Cette fonction peut signaler une erreur au cas à la communication avec un module Yoctopuce ne se passerait pas comme attendu.

### Paramètres :

**errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## YAPI.InitAPI() yInitAPI(YAPI.InitAPI())

YAPI

Initialise la librairie de programmation de Yoctopuce explicitement.

```
def InitAPI( mode, errmsg=None)
```

Il n'est pas indispensable d'appeler `yInitAPI()`, la librairie sera automatiquement initialisée de toute manière au premier appel à `yRegisterHub()`.

Lorsque cette fonction est utilisée avec comme `mode` la valeur `Y_DETECT_NONE`, il faut explicitement appeler `yRegisterHub()` pour indiquer à la librairie sur quel VirtualHub les modules sont connectés, avant d'essayer d'y accéder.

### Paramètres :

- mode** un entier spécifiant le type de détection automatique de modules à utiliser. Les valeurs possibles sont `Y_DETECT_NONE`, `Y_DETECT_USB`, `Y_DETECT_NET` et `Y_DETECT_ALL`.
- errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

### Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



## YAPI.PreregisterHub() yPreregisterHub()YAPI.PreregisterHub()

YAPI

Alternative plus tolérante à RegisterHub().

```
def PreregisterHub( url, errmsg=None)
```

Cette fonction a le même but et la même paramètres que la fonction RegisterHub, mais contrairement à celle-ci PreregisterHub() ne déclenche pas d'erreur si le hub choisi n'est pas joignable au moment de l'appel. Il est ainsi possible d'enregistrer un hub réseau indépendamment de la connectivité, afin de tenter de ne le contacter que lorsqu'on cherche réellement un module.

### Paramètres :

- url** une chaîne de caractères contenant "usb", "callback", ou l'URL racine du VirtualHub à utiliser.
- errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**YAPI.RegisterDeviceArrivalCallback()**  
**yRegisterDeviceArrivalCallback()**  
**YAPI.RegisterDeviceArrivalCallback()**

---

**YAPI**

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est branché.

```
def RegisterDeviceArrivalCallback( arrivalCallback)
```

Le callback sera appelé pendant l'exécution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

**Paramètres :**

**arrivalCallback** une procédure qui prend un `YModule` en paramètre, ou `null`

---

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

---

**YAPI**

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est débranché.

```
def RegisterDeviceRemovalCallback( removalCallback)
```

Le callback sera appelé pendant l'exécution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

**Paramètres :**

**removalCallback** une procédure qui prend un `YModule` en paramètre, ou `null`

**YAPI.RegisterHub()****YAPI****yRegisterHub()YAPI.RegisterHub()**

Configure la librairie Yoctopuce pour utiliser les modules connectés sur une machine donnée.

```
def RegisterHub( url, errmsg=None)
```

Le premier paramètre détermine le fonctionnement de l'API, il peut prendre les valeurs suivantes:

**usb**: Si vous utilisez le mot-clé **usb**, l'API utilise les modules Yoctopuce connectés directement par USB. Certains langages comme PHP, Javascript et Java ne permettent pas un accès direct aux couches matérielles, **usb** ne marchera donc pas avec ces langages. Dans ce cas, utilisez un VirtualHub ou un YoctoHub réseau (voir ci-dessous).

**x.x.x.x** ou **hostname**: L'API utilise les modules connectés à la machine dont l'adresse IP est x.x.x.x, ou dont le nom d'hôte DNS est *hostname*. Cette machine peut être un ordinateur classique faisant tourner un VirtualHub, ou un YoctoHub avec réseau (YoctoHub-Ethernet / YoctoHub-Wireless). Si vous désirez utiliser le VirtualHub tournant sur votre machine locale, utilisez l'adresse IP 127.0.0.1.

**callback** Le mot-clé **callback** permet de faire fonctionner l'API dans un mode appelé "*callback HTTP*". C'est un mode spécial permettant, entre autres, de prendre le contrôle de modules Yoctopuce à travers un filtre NAT par l'intermédiaire d'un VirtualHub ou d'un Hub Yoctopuce. Il vous suffit de configurer le hub pour qu'il appelle votre script à intervalle régulier. Ce mode de fonctionnement n'est disponible actuellement qu'en PHP et en Node.JS.

Attention, seule une application peut fonctionner à la fois sur une machine donnée en accès direct à USB, sinon il y aurait un conflit d'accès aux modules. Cela signifie en particulier que vous devez stopper le VirtualHub avant de lancer une application utilisant l'accès direct à USB. Cette limitation peut être contournée en passant par un VirtualHub plutôt que d'utiliser directement USB.

Si vous désirez vous connecter à un Hub, virtuel ou non, sur lequel le contrôle d'accès a été activé, vous devez donner le paramètre url sous la forme:

```
http://nom:mot_de_passe@adresse:port
```

Vous pouvez appeler *RegisterHub* plusieurs fois pour vous connecter à plusieurs machines différentes.

**Paramètres :**

**url** une chaîne de caractères contenant "**usb**", "**callback**", ou l'URL racine du VirtualHub à utiliser.  
**errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

YAPI

Enregistre une fonction de callback qui est appelée chaque fois qu'un hub réseau s'annonce avec un message SSDP.

```
def RegisterHubDiscoveryCallback( hubDiscoveryCallback)
```

la fonction de callback reçoit deux chaînes de caractères en paramètre La première chaîne contient le numéro de série du hub réseau et la deuxième chaîne contient l'URL du hub. L'URL peut être passée directement en argument à la fonction `yRegisterHub`. Le callback sera appelé pendant l'exécution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

### Paramètres :

**hubDiscoveryCallback** une procédure qui prend deux chaîne de caractères en paramètre, ou `null`

## YAPI.RegisterLogFunction()

YAPI

### yRegisterLogFunction()YAPI.RegisterLogFunction()

---

Enregistre une fonction de callback qui sera appelée à chaque fois que l'API a quelque chose à dire.

```
def RegisterLogFunction( logfun)
```

Utile pour débbugger le fonctionnement de l'API.

#### Paramètres :

**logfun** une procedure qui prend une chaîne de caractère en paramètre,

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

Sélectionne manuellement l'architecture de la librairie dynamique à utiliser pour accéder à USB.

```
def SelectArchitecture( arch)
```

Par défaut, la librairie Python détecte automatiquement la version de la librairie dynamique à utiliser pour accéder au port USB. Sous Linux ARM il n'est pas possible de détecter de manière fiable si il s'agit d'une installation Soft float (armel) ou Hard float (armhf). Dans ce cas, il est donc recommandé d'appeler `SelectArchitecture()` avant tout autre appel à la librairie pour forcer l'utilisation d'une architecture spécifiée.

**Paramètres :**

**arch** une chaîne de caractère spécifiant l'architecture à utiliser. Les valeurs possibles sont "armhf", "armel", "i386", "x86\_64", "32bit", "64bit"

**Retourne :**

rien. En cas d'erreur, déclenche une exception.

## YAPI.Sleep() ySleep()YAPI.Sleep()

YAPI

Effectue une pause dans l'exécution du programme pour une durée spécifiée.

```
def Sleep( ms_duration, errmsg=None)
```

L'attente est passive, c'est-à-dire qu'elle n'occupe pas significativement le processeur, de sorte à le laisser disponible pour les autres processus fonctionnant sur la machine. Durant l'attente, la librairie va néanmoins continuer à lire périodiquement les informations en provenance des modules Yoctopuce en appelant la fonction `yHandleEvents()` afin de se maintenir à jour.

Cette fonction peut signaler une erreur au cas à la communication avec un module Yoctopuce ne se passerait pas comme attendu.

### Paramètres :

**ms\_duration** un entier correspondant à la durée de la pause, en millisecondes

**errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



## YAPI.TriggerHubDiscovery() yTriggerHubDiscovery()YAPI.TriggerHubDiscovery()

YAPI

Relance une détection des hubs réseau.

```
def TriggerHubDiscovery( errmsg=None)
```

Si une fonction de callback est enregistrée avec `yRegisterDeviceRemovalCallback` elle sera appelée à chaque hub réseau qui répondra à la détection SSDP.

### Paramètres :

**errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## YAPI.UnregisterHub()

YAPI

### yUnregisterHub()YAPI.UnregisterHub()

---

Configure la librairie Yoctopuce pour ne plus utiliser les modules connectés sur une machine préalablement enregistrer avec RegisterHub.

```
def UnregisterHub( url)
```

#### Paramètres :

**url** une chaîne de caractères contenant "**usb**" ou

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

Force une mise-à-jour de la liste des modules Yoctopuce connectés.

```
def UpdateDeviceList( errmsg=None)
```

La librairie va vérifier sur les machines ou ports USB précédemment enregistrés en utilisant la fonction `yRegisterHub` si un module a été connecté ou déconnecté, et le cas échéant appeler les fonctions de callback définies par l'utilisateur.

Cette fonction peut être appelée aussi souvent que désiré, afin de rendre l'application réactive aux événements de hot-plug.

**Paramètres :**

**errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.2. Interface de la fonction Accelerometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindAccelerometer(func)

Permet de retrouver un accéléromètre d'après un identifiant donné.

#### yFirstAccelerometer()

Commence l'énumération des accéléromètres accessibles par la librairie.

### Méthodes des objets YAccelerometer

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### accelerometer→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'accéléromètre au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### accelerometer→get\_advertisedValue()

Retourne la valeur courante de l'accéléromètre (pas plus de 6 caractères).

#### accelerometer→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### accelerometer→get\_currentValue()

Retourne la valeur actuelle de l'accélération.

#### accelerometer→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'accéléromètre.

#### accelerometer→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'accéléromètre.

#### accelerometer→get\_friendlyName()

Retourne un identifiant global de l'accéléromètre au format `NOM_MODULE . NOM_FONCTION`.

#### accelerometer→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### accelerometer→get\_functionId()

Retourne l'identifiant matériel de l'accéléromètre, sans référence au module.

#### accelerometer→get\_hardwareId()

Retourne l'identifiant matériel unique de l'accéléromètre au format `SERIAL . FUNCTIONID`.

**accelerometer→get\_highestValue()**

Retourne la valeur maximale observée pour l'accélération depuis le démarrage du module.

**accelerometer→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**accelerometer→get\_logicalName()**

Retourne le nom logique de l'accéléromètre.

**accelerometer→get\_lowestValue()**

Retourne la valeur minimale observée pour l'accélération depuis le démarrage du module.

**accelerometer→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**accelerometer→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**accelerometer→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**accelerometer→get\_unit()**

Retourne l'unité dans laquelle l'accélération est exprimée.

**accelerometer→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

**accelerometer→get\_xValue()**

Retourne la composante X de l'accélération, sous forme de nombre à virgule.

**accelerometer→get\_yValue()**

Retourne la composante Y de l'accélération, sous forme de nombre à virgule.

**accelerometer→get\_zValue()**

Retourne la composante Z de l'accélération, sous forme de nombre à virgule.

**accelerometer→isOnline()**

Vérifie si le module hébergeant l'accéléromètre est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant l'accéléromètre est joignable, sans déclencher d'erreur.

**accelerometer→load(msValidity)**

Met en cache les valeurs courantes de l'accéléromètre, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

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

Met en cache les valeurs courantes de l'accéléromètre, avec une durée de validité spécifiée.

**accelerometer→nextAccelerometer()**

Continue l'énumération des accéléromètres commencée à l'aide de yFirstAccelerometer().

**accelerometer→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

### 3. Reference

#### **accelerometer→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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

Modifie la mémoire de valeur maximale observée.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique de l'accéléromètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userData.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

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

## **YAccelerometer**

Permet de retrouver un accéléromètre d'après un identifiant donné.

```
def FindAccelerometer( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'accéléromètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAccelerometer.isOnline()` pour tester si l'accéléromètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### **Paramètres :**

**func** une chaîne de caractères qui référence l'accéléromètre sans ambiguïté

### **Retourne :**

un objet de classe `YAccelerometer` qui permet ensuite de contrôler l'accéléromètre.

**YAccelerometer.FirstAccelerometer()**  
**yFirstAccelerometer()**  
**YAccelerometer.FirstAccelerometer()**

---

**YAccelerometer**

Commence l'énumération des accéléromètres accessibles par la librairie.

```
def FirstAccelerometer( )
```

Utiliser la fonction `YAccelerometer.nextAccelerometer()` pour itérer sur les autres accéléromètres.

**Retourne :**

un pointeur sur un objet `YAccelerometer`, correspondant au premier accéléromètre accessible en ligne, ou `null` si il n'y a pas de accéléromètres disponibles.



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

## **YAccelerometer**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

### **Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'accéléromètre au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'accéléromètre (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

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

---

Retourne la valeur courante de l'accéléromètre (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'accéléromètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**accelerometer→get\_currentRawValue()**

**YAccelerometer**

**accelerometer→currentRawValue()**

**accelerometer.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

---

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

---

Retourne la valeur actuelle de l'accélération.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'accélération

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**accelerometer→get\_errorMessage()**

**YAccelerometer**

**accelerometer→errorMessage()**

**accelerometer.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'accéléromètre.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'accéléromètre.

---

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'accéléromètre.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'accéléromètre.

**accelerometer→get\_friendlyName()**

**YAccelerometer**

**accelerometer→friendlyName()**

**accelerometer.get\_friendlyName()**

---

Retourne un identifiant global de l'accéléromètre au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'accéléromètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'accéléromètre (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'accéléromètre en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.



---

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

---

**YAccelerometer**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

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

---

**YAccelerometer**

Retourne l'identifiant matériel de l'accéléromètre, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'accéléromètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

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

---

Retourne l'identifiant matériel unique de l'accéléromètre au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'accéléromètre (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'accéléromètre (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**accelerometer→get\_highestValue()**

**YAccelerometer**

**accelerometer→highestValue()**

**accelerometer.get\_highestValue()**

---

Retourne la valeur maximale observée pour l'accélération depuis le démarrage du module.

def **get\_highestValue()** ( )

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'accélération depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

---

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

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**accelerometer→get\_logicalName()**

**YAccelerometer**

**accelerometer→logicalName()**

**accelerometer.get\_logicalName()**

---

Retourne le nom logique de l'accéléromètre.

def **get\_logicalName**( )

**Retourne :**

une chaîne de caractères représentant le nom logique de l'accéléromètre. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

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

---

Retourne la valeur minimale observée pour l'accélération depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'accélération depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**accelerometer**→**get\_module()**

**YAccelerometer**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`



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

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**accelerometer→get\_reportFrequency()**

**YAccelerometer**

**accelerometer→reportFrequency()**

**accelerometer.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

def **get\_reportFrequency()** ( )

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**accelerometer**→**get\_unit()**

**YAccelerometer**

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

---

Retourne l'unité dans laquelle l'accélération est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'accélération est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

---

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

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**accelerometer**→**get\_xValue()**

**YAccelerometer**

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

---

Retourne la composante X de l'accélération, sous forme de nombre à virgule.

```
def get_xValue( )
```

**Retourne :**

une valeur numérique représentant la composante X de l'accélération, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_XVALUE_INVALID`.

---

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

---

Retourne la composante Y de l'accélération, sous forme de nombre à virgule.

```
def get_yValue( )
```

**Retourne :**

une valeur numérique représentant la composante Y de l'accélération, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_YVALUE\_INVALID.

**accelerometer**→**get\_zValue()**

**YAccelerometer**

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

---

Retourne la composante Z de l'accélération, sous forme de nombre à virgule.

```
def get_zValue( )
```

**Retourne :**

une valeur numérique représentant la composante Z de l'accélération, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_ZVALUE_INVALID`.



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

Vérifie si le module hébergeant l'accéléromètre est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'accéléromètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'accéléromètre est joignable, `false` sinon

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

Met en cache les valeurs courantes de l'accéléromètre, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**accelerometer→loadCalibrationPoints()  
accelerometer.loadCalibrationPoints()**

---

**YAccelerometer**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**accelerometer**→**nextAccelerometer()**  
**accelerometer.nextAccelerometer()**

**YAccelerometer**

---

Continue l'énumération des accéléromètres commencée à l'aide de `yFirstAccelerometer()`.

def **nextAccelerometer()** ( )

**Retourne :**

un pointeur sur un objet `YAccelerometer` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**accelerometer→registerTimedReportCallback()**  
**accelerometer.registerTimedReportCallback()**

---

**YAccelerometer**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

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

---

**YAccelerometer**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### **Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

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

**YAccelerometer**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YAccelerometer**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



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

Modifie le nom logique de l'accéléromètre.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'accéléromètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YAccelerometer**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YAccelerometer**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YAccelerometer**

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

### 3.3. Interface de la fonction AnButton

La librairie de programmation Yoctopuce permet aussi bien de mesurer l'état d'un simple bouton que de lire un potentiomètre analogique (résistance variable), comme par exemple bouton rotatif continue, une poignée de commande de gaz ou un joystick. Le module est capable de se calibrer sur les valeurs minimales et maximales du potentiomètre, et de restituer une valeur calibrée variant proportionnellement avec la position du potentiomètre, indépendant de sa résistance totale.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

#### Fonction globales

##### yFindAnButton(func)

Permet de retrouver une entrée analogique d'après un identifiant donné.

##### yFirstAnButton()

Commence l'énumération des entrées analogiques accessibles par la librairie.

#### Méthodes des objets YAnButton

##### anbutton→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'entrée analogique au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

##### anbutton→get\_advertisedValue()

Retourne la valeur courante de l'entrée analogique (pas plus de 6 caractères).

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

Permet de savoir si une procédure de calibration est actuellement en cours.

##### anbutton→get\_calibratedValue()

Retourne la valeur calibrée de l'entrée (entre 0 et 1000 inclus).

##### anbutton→get\_calibrationMax()

Retourne la valeur maximale observée durant la calibration (entre 0 et 4095 inclus).

##### anbutton→get\_calibrationMin()

Retourne la valeur minimale observée durant la calibration (entre 0 et 4095 inclus).

##### anbutton→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

##### anbutton→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

##### anbutton→get\_friendlyName()

Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

**`anbutton→get_functionId()`**

Retourne l'identifiant matériel de l'entrée analogique, sans référence au module.

**`anbutton→get_hardwareId()`**

Retourne l'identifiant matériel unique de l'entrée analogique au format `SERIAL.FUNCTIONID`.

**`anbutton→get_isPressed()`**

Retourne vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon.

**`anbutton→get_lastTimePressed()`**

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière pression observée du bouton à l'entrée (transition du contact de ouvert à fermé).

**`anbutton→get_lastTimeReleased()`**

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et le dernier relâchement observée du bouton à l'entrée (transition du contact de fermé à ouvert).

**`anbutton→get_logicalName()`**

Retourne le nom logique de l'entrée analogique.

**`anbutton→get_module()`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`anbutton→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`anbutton→get_pulseCounter()`**

Retourne la valeur du compteur d'impulsions.

**`anbutton→get_pulseTimer()`**

Retourne le timer du compteur d'impulsions (ms)

**`anbutton→get_rawValue()`**

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).

**`anbutton→get_sensitivity()`**

Retourne la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

**`anbutton→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`anbutton→isOnline()`**

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

**`anbutton→isOnline_async(callback, context)`**

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

**`anbutton→load(msValidity)`**

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

**`anbutton→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

**`anbutton→nextAnButton()`**

Continue l'énumération des entrées analogiques commencée à l'aide de `yFirstAnButton()`.

**`anbutton→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`anbutton→resetCounter()`**

réinitialise le compteur d'impulsions et son timer

**`anbutton→set_analogCalibration(newval)`**

Enclenche ou déclenche le procédure de calibration.

**`anbutton→set_calibrationMax(newval)`**

### 3. Reference

Modifie la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

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

Modifie la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

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

Modifie le nom logique de l'entrée analogique.

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

Modifie la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

**anbutton**→**set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.



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

## YAnButton

Permet de retrouver une entrée analogique d'après un identifiant donné.

```
def FindAnButton( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'entrée analogique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAnButton.isOnline()` pour tester si l'entrée analogique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'entrée analogique sans ambiguïté

### Retourne :

un objet de classe `YAnButton` qui permet ensuite de contrôler l'entrée analogique.

**YAnButton.FirstAnButton()**

**YAnButton**

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

---

Commence l'énumération des entrées analogiques accessibles par la librairie.

```
def FirstAnButton( )
```

Utiliser la fonction `YAnButton.nextAnButton( )` pour itérer sur les autres entrées analogiques.

**Retourne :**

un pointeur sur un objet `YAnButton`, correspondant à la première entrée analogique accessible en ligne, ou `null` si il n'y a pas de entrées analogiques disponibles.

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

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'entrée analogique au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'entrée analogique (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**anbutton→get\_advertisedValue()**

**YAnButton**

**anbutton→advertisedValue()**

**anbutton.get\_advertisedValue()**

---

Retourne la valeur courante de l'entrée analogique (pas plus de 6 caractères).

def **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'entrée analogique (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

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

---

**YAnButton**

Permet de savoir si une procédure de calibration est actuellement en cours.

**def get\_analogCalibration( )**

**Retourne :**

soit Y\_ANALOGCALIBRATION\_OFF, soit Y\_ANALOGCALIBRATION\_ON

En cas d'erreur, déclenche une exception ou retourne Y\_ANALOGCALIBRATION\_INVALID.

**anbutton→get\_calibratedValue()**

**YAnButton**

**anbutton→calibratedValue()**

**anbutton.get\_calibratedValue()**

---

Retourne la valeur calibrée de l'entrée (entre 0 et 1000 inclus).

```
def get_calibratedValue( )
```

**Retourne :**

un entier représentant la valeur calibrée de l'entrée (entre 0 et 1000 inclus)

En cas d'erreur, déclenche une exception ou retourne Y\_CALIBRATEDVALUE\_INVALID.

---

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

---

**YAnButton**

Retourne la valeur maximale observée durant la calibration (entre 0 et 4095 inclus).

**def get\_calibrationMax( )**

**Retourne :**

un entier représentant la valeur maximale observée durant la calibration (entre 0 et 4095 inclus)

En cas d'erreur, déclenche une exception ou retourne Y\_CALIBRATIONMAX\_INVALID.

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

---

**YAnButton**

Retourne la valeur minimale observée durant la calibration (entre 0 et 4095 inclus).

```
def get_calibrationMin( )
```

**Retourne :**

un entier représentant la valeur minimale observée durant la calibration (entre 0 et 4095 inclus)

En cas d'erreur, déclenche une exception ou retourne Y\_CALIBRATIONMIN\_INVALID.



---

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'entrée analogique.

**anbutton→get\_errorType()**

**YAnButton**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'entrée analogique.

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

Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'entrée analogique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'entrée analogique (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'entrée analogique en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YAnButton**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

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

---

Retourne l'identifiant matériel de l'entrée analogique, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'entrée analogique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**anbutton**→**get\_hardwareId()**

**YAnButton**

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

---

Retourne l'identifiant matériel unique de l'entrée analogique au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'entrée analogique (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'entrée analogique (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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

---

Retourne vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon.

```
def get_isPressed( )
```

**Retourne :**

soit Y\_ISPRESSED\_FALSE, soit Y\_ISPRESSED\_TRUE, selon vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon

En cas d'erreur, déclenche une exception ou retourne Y\_ISPRESSED\_INVALID.

**anbutton→get\_lastTimePressed()**

**YAnButton**

**anbutton→lastTimePressed()**

**anbutton.get\_lastTimePressed()**

---

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière pression observée du bouton à l'entrée (transition du contact de ouvert à fermé).

```
def get_lastTimePressed( )
```

**Retourne :**

un entier représentant le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière pression observée du bouton à l'entrée (transition du contact de ouvert à fermé)

En cas d'erreur, déclenche une exception ou retourne Y\_LASTTIMEPRESSED\_INVALID.



---

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

---

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et le dernier relâchement observée du bouton à l'entrée (transition du contact de fermé à ouvert).

```
def get_lastTimeReleased( )
```

**Retourne :**

un entier représentant le temps absolu (nombre de millisecondes) entre la mise sous tension du module et le dernier relâchement observée du bouton à l'entrée (transition du contact de fermé à ouvert)

En cas d'erreur, déclenche une exception ou retourne Y\_LASTTIMERELEASED\_INVALID.

**anbutton**→**get\_logicalName()**

**YAnButton**

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

---

Retourne le nom logique de l'entrée analogique.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'entrée analogique. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**anbutton→get\_pulseCounter()**

**YAnButton**

**anbutton→pulseCounter()**

**anbutton.get\_pulseCounter()**

---

Retourne la valeur du compteur d'impulsions.

```
def get_pulseCounter( )
```

**Retourne :**

un entier représentant la valeur du compteur d'impulsions

En cas d'erreur, déclenche une exception ou retourne Y\_PULSECOUNTER\_INVALID.

---

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

---

Retourne le timer du compteur d'impulsions (ms)

```
def get_pulseTimer( )
```

**Retourne :**

un entier représentant le timer du compteur d'impulsions (ms)

En cas d'erreur, déclenche une exception ou retourne Y\_PULSETIMER\_INVALID.

**anbutton**→**get\_rawValue()**

**YAnButton**

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

---

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).

```
def get_rawValue( )
```

**Retourne :**

un entier représentant la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus)

En cas d'erreur, déclenche une exception ou retourne `Y_RAWVALUE_INVALID`.

---

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

---

Retourne la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

```
def get_sensitivity( )
```

**Retourne :**

un entier représentant la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks

En cas d'erreur, déclenche une exception ou retourne `Y_SENSITIVITY_INVALID`.

**anbutton**→**get\_userData()**

**YAnButton**

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

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.



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

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'entrée analogique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'entrée analogique est joignable, `false` sinon

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

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

Continue l'énumération des entrées analogiques commencée à l'aide de `yFirstAnButton()`.

```
def nextAnButton( )
```

**Retourne :**

un pointeur sur un objet `YAnButton` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**anbutton→registerValueCallback()**  
**anbutton.registerValueCallback()****YAnButton**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

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

---

réinitialise le compteur d'impulsions et son timer

```
def resetCounter( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YAnButton**

Enclenche ou déclenche le procédure de calibration.

```
def set_analogCalibration( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module à la fin de la calibration si le réglage doit être préservé.

**Paramètres :**

**newval** soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

**YAnButton**

Modifie la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

```
def set_calibrationMax( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YAnButton**

---

Modifie la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

```
def set_calibrationMin( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

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

---

**YAnButton**

Modifie le nom logique de l'entrée analogique.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'entrée analogique.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Modifie la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

```
def set_sensitivity( newval)
```

La sensibilité sert à filtrer les variations autour d'une valeur fixe, mais ne préterite pas la transmission d'événements lorsque la valeur d'entrée évolue constamment dans la même direction. Cas particulier: lorsque la valeur 1000 est utilisée, seuls les valeurs déclenchant une commutation d'état pressé/non-pressé sont transmises. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
def set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.4. Interface de la fonction CarbonDioxide

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindCarbonDioxide(func)

Permet de retrouver un capteur de CO2 d'après un identifiant donné.

#### yFirstCarbonDioxide()

Commence l'énumération des capteurs de CO2 accessibles par la librairie.

### Méthodes des objets YCarbonDioxide

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### carbondioxide→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de CO2 au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### carbondioxide→get\_advertisedValue()

Retourne la valeur courante du capteur de CO2 (pas plus de 6 caractères).

#### carbondioxide→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### carbondioxide→get\_currentValue()

Retourne la valeur actuelle du taux de CO2.

#### carbondioxide→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO2.

#### carbondioxide→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO2.

#### carbondioxide→get\_friendlyName()

Retourne un identifiant global du capteur de CO2 au format `NOM_MODULE . NOM_FONCTION`.

#### carbondioxide→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### carbondioxide→get\_functionId()

Retourne l'identifiant matériel du capteur de CO2, sans référence au module.

#### carbondioxide→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de CO2 au format `SERIAL . FUNCTIONID`.

**carbondioxide→get\_highestValue()**

Retourne la valeur maximale observée pour le taux de CO2 depuis le démarrage du module.

**carbondioxide→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**carbondioxide→get\_logicalName()**

Retourne le nom logique du capteur de CO2.

**carbondioxide→get\_lowestValue()**

Retourne la valeur minimale observée pour le taux de CO2 depuis le démarrage du module.

**carbondioxide→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**carbondioxide→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**carbondioxide→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**carbondioxide→get\_unit()**

Retourne l'unité dans laquelle le taux de CO2 est exprimée.

**carbondioxide→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

**carbondioxide→isOnline()**

Vérifie si le module hébergeant le capteur de CO2 est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le capteur de CO2 est joignable, sans déclencher d'erreur.

**carbondioxide→load(msValidity)**

Met en cache les valeurs courantes du capteur de CO2, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

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

Met en cache les valeurs courantes du capteur de CO2, avec une durée de validité spécifiée.

**carbondioxide→nextCarbonDioxide()**

Continue l'énumération des capteurs de CO2 commencée à l'aide de yFirstCarbonDioxide().

**carbondioxide→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**carbondioxide→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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

Modifie la mémoire de valeur maximale observée.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

### 3. Reference

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

Modifie le nom logique du capteur de CO2.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

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

## **YCarbonDioxide**

Permet de retrouver un capteur de CO2 d'après un identifiant donné.

```
def FindCarbonDioxide( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de CO2 soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCarbonDioxide.isOnline()` pour tester si le capteur de CO2 est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### **Paramètres :**

**func** une chaîne de caractères qui référence le capteur de CO2 sans ambiguïté

### **Retourne :**

un objet de classe `YCarbonDioxide` qui permet ensuite de contrôler le capteur de CO2.

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

---

**YCarbonDioxide**

Commence l'énumération des capteurs de CO2 accessibles par la librairie.

```
def FirstCarbonDioxide( )
```

Utiliser la fonction `YCarbonDioxide.nextCarbonDioxide()` pour itérer sur les autres capteurs de CO2.

**Retourne :**

un pointeur sur un objet `YCarbonDioxide`, correspondant au premier capteur de CO2 accessible en ligne, ou `null` si il n'y a pas de capteurs de CO2 disponibles.



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

## **YCarbonDioxide**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

### **Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de CO2 au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de CO2 (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

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

---

Retourne la valeur courante du capteur de CO2 (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de CO2 (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

carbondioxide→get\_currentRawValue()

YCarbonDioxide

carbondioxide→currentRawValue()

carbondioxide.get\_currentRawValue()

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

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

**YCarbonDioxide**

Retourne la valeur actuelle du taux de CO2.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du taux de CO2

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

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

**YCarbonDioxide**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO2.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de CO2.

---

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

---

**YCarbonDioxide**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO2.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de CO2.

**carbondioxide→get\_friendlyName()**

**YCarbonDioxide**

**carbondioxide→friendlyName()**

**carbondioxide.get\_friendlyName()**

---

Retourne un identifiant global du capteur de CO2 au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de CO2 si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de CO2 (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de CO2 en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.



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

---

**YCarbonDioxide**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

---

**YCarbonDioxide**

Retourne l'identifiant matériel du capteur de CO2, sans référence au module.

```
def get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le capteur de CO2 (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

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

**YCarbonDioxide**

---

Retourne l'identifiant matériel unique du capteur de CO2 au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de CO2 (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de CO2 (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

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

**YCarbonDioxide**

---

Retourne la valeur maximale observée pour le taux de CO2 depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le taux de CO2 depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

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

**YCarbonDioxide**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**carbondioxide→get\_logicalName()**

**YCarbonDioxide**

**carbondioxide→logicalName()**

**carbondioxide.get\_logicalName()**

---

Retourne le nom logique du capteur de CO2.

def **get\_logicalName**( )

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de CO2. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

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

---

Retourne la valeur minimale observée pour le taux de CO2 depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le taux de CO2 depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

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

---

**YCarbonDioxide**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule



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

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**carbondioxide→get\_reportFrequency()**

**YCarbonDioxide**

**carbondioxide→reportFrequency()**

**carbondioxide.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**carbondioxide**→**get\_unit()**

**YCarbonDioxide**

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

---

Retourne l'unité dans laquelle le taux de CO2 est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le taux de CO2 est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

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

**YCarbonDioxide**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

Vérifie si le module hébergeant le capteur de CO2 est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache du capteur de CO2 sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de CO2 est joignable, false sinon

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

Met en cache les valeurs courantes du capteur de CO2, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**carbondioxide→loadCalibrationPoints()**  
**carbondioxide.loadCalibrationPoints()**

**YCarbonDioxide**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**carbondioxide→nextCarbonDioxide()**  
**carbondioxide.nextCarbonDioxide()****YCarbonDioxide**

---

Continue l'énumération des capteurs de CO2 commencée à l'aide de `yFirstCarbonDioxide()`.

def **nextCarbonDioxide()**

**Retourne :**

un pointeur sur un objet `YCarbonDioxide` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**carbondioxide→registerTimedReportCallback()  
carbondioxide.registerTimedReportCallback()****YCarbonDioxide**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

**carbondioxide→registerValueCallback()**  
**carbondioxide.registerValueCallback()**

---

**YCarbonDioxide**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

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

---

**YCarbonDioxide**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YCarbonDioxide**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YCarbonDioxide**

Modifie le nom logique du capteur de CO2.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de CO2.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YCarbonDioxide**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**carbondioxide→set\_reportFrequency()**  
**carbondioxide→setReportFrequency()**  
**carbondioxide.set\_reportFrequency()**

**YCarbonDioxide**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



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

**YCarbonDioxide**

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**carbondioxide→set\_userdata()**  
**carbondioxide→setUserData()**  
**carbondioxide.set\_userdata()**

**YCarbonDioxide**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

```
def set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.5. Interface de la fonction ColorLed

La librairie de programmation Yoctopuce permet de piloter une led couleur aussi bien en coordonnées RGB qu'en coordonnées HSL, les conversions RGB vers HSL étant faites automatiquement par le module. Ceci permet aisément d'allumer la led avec une certaine teinte et d'en faire progressivement varier la saturation ou la luminosité. Si nécessaire, vous trouverez plus d'information sur la différence entre RGB et HSL dans la section suivante.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### **yFindColorLed(func)**

Permet de retrouver une led RGB d'après un identifiant donné.

#### **yFirstColorLed()**

Commence l'énumération des leds RGB accessibles par la librairie.

### Méthodes des objets YColorLed

#### **colorled→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led RGB au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

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

Retourne la valeur courante de la led RGB (pas plus de 6 caractères).

#### **colorled→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led RGB.

#### **colorled→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led RGB.

#### **colorled→get\_friendlyName()**

Retourne un identifiant global de la led RGB au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **colorled→get\_functionId()**

Retourne l'identifiant matériel de la led RGB, sans référence au module.

#### **colorled→get\_hardwareId()**

Retourne l'identifiant matériel unique de la led RGB au format `SERIAL . FUNCTIONID`.

#### **colorled→get\_hslColor()**

Retourne la couleur HSL courante de la led.

#### **colorled→get\_logicalName()**

Retourne le nom logique de la led RGB.

**colorled→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**colorled→get\_rgbColor()**

Retourne la couleur RGB courante de la led.

**colorled→get\_rgbColorAtPowerOn()**

Retourne la couleur configurée pour être affichée à l'allumage du module.

**colorled→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**colorled→hslMove(hsl\_target, ms\_duration)**

Effectue une transition continue dans l'espace HSL entre la couleur courante et une nouvelle couleur.

**colorled→isOnline()**

Vérifie si le module hébergeant la led RGB est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant la led RGB est joignable, sans déclencher d'erreur.

**colorled→load(msValidity)**

Met en cache les valeurs courantes de la led RGB, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes de la led RGB, avec une durée de validité spécifiée.

**colorled→nextColorLed()**

Continue l'énumération des leds RGB commencée à l'aide de `yFirstColorLed()`.

**colorled→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**colorled→rgbMove(rgb\_target, ms\_duration)**

Effectue une transition continue dans l'espace RGB entre la couleur courante et une nouvelle couleur.

**colorled→set\_hslColor(newval)**

Modifie la couleur courante de la led, en utilisant une couleur HSL spécifiée.

**colorled→set\_logicalName(newval)**

Modifie le nom logique de la led RGB.

**colorled→set\_rgbColor(newval)**

Modifie la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu).

**colorled→set\_rgbColorAtPowerOn(newval)**

Modifie la couleur que la led va afficher spontanément à l'allumage du module.

**colorled→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YColorLed.FindColorLed() yFindColorLed()YColorLed.FindColorLed()

## YColorLed

Permet de retrouver une led RGB d'après un identifiant donné.

```
def FindColorLed( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la led RGB soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YColorLed.isOnline()` pour tester si la led RGB est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence la led RGB sans ambiguïté

### Retourne :

un objet de classe `YColorLed` qui permet ensuite de contrôler la led RGB.

**YColorLed.FirstColorLed()**

**YColorLed**

**yFirstColorLed()YColorLed.FirstColorLed()**

---

Commence l'énumération des leds RGB accessibles par la librairie.

```
def FirstColorLed( )
```

Utiliser la fonction `YColorLed.nextColorLed( )` pour itérer sur les autres leds RGB.

**Retourne :**

un pointeur sur un objet `YColorLed`, correspondant à la première led RGB accessible en ligne, ou `null` si il n'y a pas de leds RGB disponibles.

**colorled→describe()colorled.describe()****YColorLed**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led RGB au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant la led RGB (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

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

**YColorLed**

**colorled**→**advertisedValue()**

**colorled.get\_advertisedValue()**

---

Retourne la valeur courante de la led RGB (pas plus de 6 caractères).

def **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante de la led RGB (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.



---

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

---

**YColorLed**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led RGB.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la led RGB.

**colorled**→**get\_errorType()**

**YColorLed**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led RGB.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la led RGB.

---

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

---

Retourne un identifiant global de la led RGB au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la led RGB si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la led RGB (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant la led RGB en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YColorLed**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**colorled**→**get\_functionId()****YColorLed****colorled**→**functionId()****colorled.get\_functionId()**

---

Retourne l'identifiant matériel de la led RGB, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la led RGB (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**colorled**→**get\_hardwareId()**

**YColorLed**

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

---

Retourne l'identifiant matériel unique de la led RGB au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led RGB (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant la led RGB (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**colorled**→**get\_hslColor()****YColorLed****colorled**→**hslColor()****colorled.get\_hslColor()**

---

Retourne la couleur HSL courante de la led.

```
def get_hslColor( )
```

**Retourne :**

un entier représentant la couleur HSL courante de la led

En cas d'erreur, déclenche une exception ou retourne `Y_HSLCOLOR_INVALID`.

**colorled**→**get\_logicalName()**

**YColorLed**

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

---

Retourne le nom logique de la led RGB.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de la led RGB. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.



---

**colorled**→**get\_module()****YColorLed****colorled**→**module()****colorled.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**colorled**→**get\_rgbColor()**

**YColorLed**

**colorled**→**rgbColor()****colorled.get\_rgbColor()**

---

Retourne la couleur RGB courante de la led.

```
def get_rgbColor( )
```

**Retourne :**

un entier représentant la couleur RGB courante de la led

En cas d'erreur, déclenche une exception ou retourne `Y_RGBCOLOR_INVALID`.

---

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

---

Retourne la couleur configurée pour être affichage à l'allumage du module.

```
def get_rgbColorAtPowerOn( )
```

**Retourne :**

un entier représentant la couleur configurée pour être affichage à l'allumage du module

En cas d'erreur, déclenche une exception ou retourne Y\_RGBCOLORATPOWERON\_INVALID.

**colorled**→**get\_userdata()**

**YColorLed**

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

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

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

Effectue une transition continue dans l'espace HSL entre la couleur courante et une nouvelle couleur.

```
def hslMove( hsl_target, ms_duration)
```

**Paramètres :**

**hsl\_target** couleur HSL désirée à la fin de la transition

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **colorled**→**isOnline()****colorled.isOnline()**

**YColorLed**

Vérifie si le module hébergeant la led RGB est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de la led RGB sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si la led RGB est joignable, `false` sinon

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

Met en cache les valeurs courantes de la led RGB, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**colorled**→**nextColorLed()****colorled.nextColorLed()**

**YColorLed**

---

Continue l'énumération des leds RGB commencée à l'aide de `yFirstColorLed()`.

def **nextColorLed**( )

**Retourne :**

un pointeur sur un objet `YColorLed` accessible en ligne, ou `null` lorsque l'énumération est terminée.



---

**colorled→registerValueCallback()**  
**colorled.registerValueCallback()**

---

**YColorLed**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## colorled→rgbMove()colorled.rgbMove()

YColorLed

Effectue une transition continue dans l'espace RGB entre la couleur courante et une nouvelle couleur.

```
def rgbMove( rgb_target, ms_duration)
```

### Paramètres :

**rgb\_target** couleur RGB désirée à la fin de la transition

**ms\_duration** durée de la transition, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

Modifie la couleur courante de la led, en utilisant une couleur HSL spécifiée.

```
def set_hslColor( newval)
```

L'encodage est réalisé de la manière suivante: 0xHHSSL.

**Paramètres :**

**newval** un entier représentant la couleur courante de la led, en utilisant une couleur HSL spécifiée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**colorled**→**set\_logicalName()**

**YColorLed**

**colorled**→**setLogicalName()**

**colorled.set\_logicalName()**

---

Modifie le nom logique de la led RGB.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la led RGB.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

Modifie la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu).

```
def set_rgbColor( newval)
```

L'encodage est réalisé de la manière suivante: 0xRRGGBB.

**Paramètres :**

**newval** un entier représentant la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**colorled**→**set\_rgbColorAtPowerOn()**  
**colorled**→**setRgbColorAtPowerOn()**  
**colorled.set\_rgbColorAtPowerOn()**

---

**YColorLed**

Modifie la couleur que la led va afficher spontanément à l'allumage du module.

```
def set_rgbColorAtPowerOn( newval)
```

Cette couleur sera affichée dès que le module sera sous tension. Ne pas oublier d'appeler la fonction `saveToFlash()` du module correspondant pour que ce paramètre soit mémorisé.

**Paramètres :**

**newval** un entier représentant la couleur que la led va afficher spontanément à l'allumage du module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**colorled**→**set\_userdata()****YColorLed****colorled**→**setUserData()****colorled.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.6. Interface de la fonction Compass

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindCompass(func)

Permet de retrouver un compas d'après un identifiant donné.

#### yFirstCompass()

Commence l'énumération des compas accessibles par la librairie.

### Méthodes des objets YCompass

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### compass→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du compas au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### compass→get\_advertisedValue()

Retourne la valeur courante du compas (pas plus de 6 caractères).

#### compass→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### compass→get\_currentValue()

Retourne la valeur actuelle du cap relatif.

#### compass→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du compas.

#### compass→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du compas.

#### compass→get\_friendlyName()

Retourne un identifiant global du compas au format `NOM_MODULE . NOM_FONCTION`.

#### compass→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### compass→get\_functionId()

Retourne l'identifiant matériel du compas, sans référence au module.

#### compass→get\_hardwareId()

Retourne l'identifiant matériel unique du compas au format `SERIAL . FUNCTIONID`.



**compass→get\_highestValue()**

Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.

**compass→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**compass→get\_logicalName()**

Retourne le nom logique du compas.

**compass→get\_lowestValue()**

Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.

**compass→get\_magneticHeading()**

Retourne la direction du nord magnétique, indépendamment du cap configuré.

**compass→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**compass→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**compass→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**compass→get\_unit()**

Retourne l'unité dans laquelle le cap relatif est exprimée.

**compass→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**compass→isOnline()**

Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.

**compass→load(msValidity)**

Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

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

Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.

**compass→nextCompass()**

Continue l'énumération des compas commencée à l'aide de `yFirstCompass()`.

**compass→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**compass→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**compass→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

### 3. Reference

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du compas.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YCompass.FindCompass() yFindCompass()YCompass.FindCompass()

## YCompass

Permet de retrouver un compas d'après un identifiant donné.

```
def FindCompass( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le compas soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCompass.isOnline()` pour tester si le compas est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le compas sans ambiguïté

### Retourne :

un objet de classe `YCompass` qui permet ensuite de contrôler le compas.

**YCompass.FirstCompass()**

**YCompass**

**yFirstCompass()YCompass.FirstCompass()**

---

Commence l'énumération des compas accessibles par la librairie.

```
def FirstCompass( )
```

Utiliser la fonction `YCompass.nextCompass()` pour itérer sur les autres compas.

**Retourne :**

un pointeur sur un objet `YCompass`, correspondant au premier compas accessible en ligne, ou `null` si il n'y a pas de compas disponibles.

**compass→calibrateFromPoints()**  
**compass.calibrateFromPoints()****YCompass**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**compass→describe()compass.describe()****YCompass**

Retourne un court texte décrivant de manière non-ambigüe l'instance du compas au format  
`TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le compas (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

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

---

Retourne la valeur courante du compas (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du compas (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

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

---

**YCompass**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.



---

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

---

**YCompass**

Retourne la valeur actuelle du cap relatif.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du cap relatif

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

**compass→get\_errorMessage()**

**YCompass**

**compass→errorMessage()**

**compass.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du compas.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du compas.

---

**compass**→**get\_errorType()****YCompass****compass**→**errorType()****compass.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du compas.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du compas.

**compass**→**get\_friendlyName()**

**YCompass**

**compass**→**friendlyName()**

**compass.get\_friendlyName()**

---

Retourne un identifiant global du compas au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du compas si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du compas (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le compas en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

---

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

---

**YCompass**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**compass**→**get\_functionId()**

**YCompass**

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

---

Retourne l'identifiant matériel du compas, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le compas (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

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

Retourne l'identifiant matériel unique du compas au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du compas (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le compas (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**compass→get\_highestValue()**

**YCompass**

**compass→highestValue()**

**compass.get\_highestValue()**

---

Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le cap relatif depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.



---

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

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**compass**→**get\_logicalName()**

**YCompass**

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

---

Retourne le nom logique du compas.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du compas. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

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

---

Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le cap relatif depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

**compass**→**get\_magneticHeading()**

**YCompass**

**compass**→**magneticHeading()**

**compass.get\_magneticHeading()**

---

Retourne la direction du nord magnétique, indépendamment du cap configuré.

```
def get_magneticHeading( )
```

**Retourne :**

une valeur numérique représentant la direction du nord magnétique, indépendamment du cap configuré

En cas d'erreur, déclenche une exception ou retourne Y\_MAGNETICHEADING\_INVALID.

---

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

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

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**compass→get\_reportFrequency()****YCompass****compass→reportFrequency()****compass.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**compass**→**get\_resolution()**

**YCompass**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.



**compass**→**get\_unit()****YCompass****compass**→**unit()****compass.get\_unit()**

Retourne l'unité dans laquelle le cap relatif est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le cap relatif est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**compass**→**get\_userData()**

**YCompass**

**compass**→**userData()****compass.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**compass**→**isOnline()****compass.isOnline()****YCompass**

Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du compas sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le compas est joignable, `false` sinon

**compass→load()compass.load()****YCompass**

Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**compass→loadCalibrationPoints()**  
**compass.loadCalibrationPoints()**

---

**YCompass**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**compass**→**nextCompass()****compass.nextCompass()**

**YCompass**

---

Continue l'énumération des compas commencée à l'aide de `yFirstCompass()`.

def **nextCompass**( )

**Retourne :**

un pointeur sur un objet `YCompass` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**compass→registerTimedReportCallback()**  
**compass.registerTimedReportCallback()**

---

**YCompass**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**compass→registerValueCallback()**  
**compass.registerValueCallback()****YCompass**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.



---

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

---

**YCompass**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YCompass**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

**YCompass**

Modifie le nom logique du compas.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du compas.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YCompass**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**compass→set\_reportFrequency()**  
**compass→setReportFrequency()**  
**compass.set\_reportFrequency()**

**YCompass**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**compass**→**set\_resolution()**

**YCompass**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**compass**→**set\_userData()****YCompass****compass**→**setUserData()****compass.set\_userData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
def set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.7. Interface de la fonction Current

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindCurrent(func)

Permet de retrouver un capteur de courant d'après un identifiant donné.

#### yFirstCurrent()

Commence l'énumération des capteurs de courant accessibles par la librairie.

### Méthodes des objets YCurrent

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### current→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### current→get\_advertisedValue()

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

#### current→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### current→get\_currentValue()

Retourne la valeur instantanée du courant.

#### current→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

#### current→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

#### current→get\_friendlyName()

Retourne un identifiant global du capteur de courant au format `NOM_MODULE . NOM_FONCTION`.

#### current→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### current→get\_functionId()

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

#### current→get\_hardwareId()



Retourne l'identifiant matériel unique du capteur de courant au format `SERIAL.FUNCTIONID`.

**current→get\_highestValue()**

Retourne la valeur maximale observée pour le courant.

**current→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**current→get\_logicalName()**

Retourne le nom logique du capteur de courant.

**current→get\_lowestValue()**

Retourne la valeur minimale observée pour le courant.

**current→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**current→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**current→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**current→get\_unit()**

Retourne l'unité dans laquelle le courant est exprimée.

**current→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**current→isOnline()**

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

**current→load(msValidity)**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

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

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

**current→nextCurrent()**

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

**current→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**current→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**current→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour le courant.

**current→set\_logFrequency(newval)**

### 3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur de courant.

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

Modifie la mémoire de valeur minimale observée pour le courant.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs mesurées.

**current**→**set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## **YCurrent.FindCurrent()** **yFindCurrent()YCurrent.FindCurrent()**

**YCurrent**

Permet de retrouver un capteur de courant d'après un identifiant donné.

```
def FindCurrent( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de courant soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de courant sans ambiguïté

**Retourne :**

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

**YCurrent.FirstCurrent()**

**YCurrent**

**yFirstCurrent()****YCurrent.FirstCurrent()**

---

Commence l'énumération des capteurs de courant accessibles par la librairie.

```
def FirstCurrent( )
```

Utiliser la fonction `YCurrent.nextCurrent( )` pour itérer sur les autres capteurs de courant.

**Retourne :**

un pointeur sur un objet `YCurrent`, correspondant au premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

**current→calibrateFromPoints()  
current.calibrateFromPoints()****YCurrent**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→describe()current.describe()****YCurrent**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de courant (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

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

---

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de courant (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**current**→**get\_currentRawValue()**

**YCurrent**

**current**→**currentRawValue()**

**current.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.



---

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

---

Retourne la valeur instantanée du courant.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur instantanée du courant

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**current**→**get\_errorMessage()**

**YCurrent**

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

---

**current**→**get\_errorType()****YCurrent****current**→**errorType()****current.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

**current**→**get\_friendlyName()**

**YCurrent**

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

---

Retourne un identifiant global du capteur de courant au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de courant si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de courant (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de courant en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

---

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

---

**YCurrent**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**current**→**get\_functionId()**

**YCurrent**

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

---

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de courant (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

**current**→**get\_hardwareId()****YCurrent****current**→**hardwareId()****current.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du capteur de courant au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de courant (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de courant (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**current**→**get\_highestValue()**

**YCurrent**

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

---

Retourne la valeur maximale observée pour le courant.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le courant

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.



---

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

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**current**→**get\_logicalName()**

**YCurrent**

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

---

Retourne le nom logique du capteur de courant.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de courant. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

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

---

Retourne la valeur minimale observée pour le courant.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le courant

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

**current**→**get\_module()**

**YCurrent**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

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

---

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**current→get\_reportFrequency()**

**YCurrent**

**current→reportFrequency()**

**current.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

def **get\_reportFrequency**( )

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

**current**→**get\_resolution()****YCurrent****current**→**resolution()****current.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

**current**→**get\_unit()**

**YCurrent**

**current**→**unit()****current.get\_unit()**

---

Retourne l'unité dans laquelle le courant est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le courant est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.



---

**current**→**get\_userData()****YCurrent****current**→**userData()****current.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**current**→**isOnline()****current.isOnline()**

**YCurrent**

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de courant est joignable, `false` sinon

**current→load()current.load()****YCurrent**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→loadCalibrationPoints()**

**YCurrent**

**current.loadCalibrationPoints()**

---

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**current**→**nextCurrent()****current.nextCurrent()****YCurrent**

---

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

`def nextCurrent()`

**Retourne :**

un pointeur sur un objet `YCurrent` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**current→registerTimedReportCallback()  
current.registerTimedReportCallback()****YCurrent**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

**current→registerValueCallback()**  
**current.registerValueCallback()**

---

**YCurrent**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**current**→**set\_highestValue()**

**YCurrent**

**current**→**setHighestValue()**

**current.set\_highestValue()**

---

Modifie la mémoire de valeur maximale observée pour le courant.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour le courant

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

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

---

**YCurrent**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current**→**set\_logicalName()**

**YCurrent**

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

---

Modifie le nom logique du capteur de courant.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de courant.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current**→**set\_lowestValue()****YCurrent****current**→**setLowestValue()****current.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée pour le courant.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour le courant

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current**→**set\_reportFrequency()**  
**current**→**setReportFrequency()**  
**current.set\_reportFrequency()**

---

**YCurrent**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

Modifie la résolution des valeurs mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current**→**set\_userdata()**

**YCurrent**

**current**→**setUserData()****current.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.8. Interface de la fonction DataLogger

Les capteurs de Yoctopuce sont équipés d'une mémoire non-volatile permettant de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La fonction DataLogger contrôle les paramètres globaux de cet enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindDataLogger(func)

Permet de retrouver un enregistreur de données d'après un identifiant donné.

#### yFirstDataLogger()

Commence l'énumération des enregistreurs de données accessibles par la librairie.

### Méthodes des objets YDataLogger

#### datalogger→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### datalogger→forgetAllDataStreams()

Efface tout l'historique des mesures de l'enregistreur de données.

#### datalogger→get\_advertisedValue()

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

#### datalogger→get\_autoStart()

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

#### datalogger→get\_currentRunIndex()

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

#### datalogger→get\_dataSets()

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

#### datalogger→get\_dataStreams(v)

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

#### datalogger→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

#### datalogger→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

#### datalogger→get\_friendlyName()

	Retourne un identifiant global de l'enregistreur de données au format <code>NOM_MODULE . NOM_FONCTION</code> .
<b><code>datalogger→get_functionDescriptor()</code></b>	Retourne un identifiant unique de type <code>YFUN_DESCR</code> correspondant à la fonction.
<b><code>datalogger→get_functionId()</code></b>	Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.
<b><code>datalogger→get_hardwareId()</code></b>	Retourne l'identifiant matériel unique de l'enregistreur de données au format <code>SERIAL . FUNCTIONID</code> .
<b><code>datalogger→get_logicalName()</code></b>	Retourne le nom logique de l'enregistreur de données.
<b><code>datalogger→get_module()</code></b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b><code>datalogger→get_module_async(callback, context)</code></b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b><code>datalogger→get_recording()</code></b>	Retourne l'état d'activation de l'enregistreur de données.
<b><code>datalogger→get_timeUTC()</code></b>	Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.
<b><code>datalogger→get_userData()</code></b>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<b><code>datalogger→isOnline()</code></b>	Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.
<b><code>datalogger→isOnline_async(callback, context)</code></b>	Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.
<b><code>datalogger→load(msValidity)</code></b>	Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.
<b><code>datalogger→load_async(msValidity, callback, context)</code></b>	Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.
<b><code>datalogger→nextDataLogger()</code></b>	Continue l'énumération des enregistreurs de données commencée à l'aide de <code>yFirstDataLogger()</code> .
<b><code>datalogger→registerValueCallback(callback)</code></b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b><code>datalogger→set_autoStart(newval)</code></b>	Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.
<b><code>datalogger→set_logicalName(newval)</code></b>	Modifie le nom logique de l'enregistreur de données.
<b><code>datalogger→set_recording(newval)</code></b>	Modifie l'état d'activation de l'enregistreur de données.
<b><code>datalogger→set_timeUTC(newval)</code></b>	Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.
<b><code>datalogger→set_userData(data)</code></b>	Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
<b><code>datalogger→wait_async(callback, context)</code></b>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.



**YDataLogger.FindDataLogger()****YDataLogger****yFindDataLogger()YDataLogger.FindDataLogger()**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

```
def FindDataLogger( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'enregistreur de données soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnline()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

**Retourne :**

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

## YDataLogger.FirstDataLogger()

YDataLogger

## yFirstDataLogger()YDataLogger.FirstDataLogger()

---

Commence l'énumération des enregistreurs de données accessibles par la librairie.

```
def FirstDataLogger( )
```

Utiliser la fonction `YDataLogger.nextDataLogger( )` pour itérer sur les autres enregistreurs de données.

### Retourne :

un pointeur sur un objet `YDataLogger`, correspondant au premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

**datalogger→describe()****datalogger.describe()****YDataLogger**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'enregistreur de données (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**datalogger→forgetAllDataStreams()**  
**datalogger.forgetAllDataStreams()**

---

**YDataLogger**

Efface tout l'historique des mesures de l'enregistreur de données.

```
def forgetAllDataStreams( )
```

Cette méthode remet aussi à zéro le compteur de Runs.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**datalogger→get\_advertisedValue()****YDataLogger****datalogger→advertisedValue()****datalogger.get\_advertisedValue()**

---

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'enregistreur de données (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**datalogger**→**get\_autoStart()**

**YDataLogger**

**datalogger**→**autoStart()****datalogger.get\_autoStart()**

---

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

```
def get_autoStart( )
```

**Retourne :**

soit Y\_AUTOSTART\_OFF, soit Y\_AUTOSTART\_ON, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

En cas d'erreur, déclenche une exception ou retourne Y\_AUTOSTART\_INVALID.

---

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

---

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

```
def get_currentRunIndex( )
```

**Retourne :**

un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRUNINDEX\_INVALID.

**datalogger**→**get\_dataSets()**

**YDataLogger**

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

---

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

```
def get_dataSets( )
```

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Retourne :**

une liste d'objets YDataSet

En cas d'erreur, déclenche une exception ou retourne une liste vide.



**datalogger→get\_dataStreams()****YDataLogger****datalogger→dataStreams()****datalogger.get\_dataStreams()**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

```
def get_dataStreams( v)
```

L'appelant doit passer par référence un tableau vide pour stocker les objets YDataStream, et la méthode va les remplir avec des objets décrivant les séquences de données disponibles.

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

**Paramètres :**

**v** un tableau de YDataStreams qui sera rempli avec les séquences trouvées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→get\_errorMessage()**

**YDataLogger**

**datalogger→errorMessage()**

**datalogger.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

**def get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

---

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

**datalogger→get\_friendlyName()**

**YDataLogger**

**datalogger→friendlyName()**

**datalogger.get\_friendlyName()**

---

Retourne un identifiant global de l'enregistreur de données au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'enregistreur de données si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'enregistreur de données (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

---

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

---

**YDataLogger**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

**datalogger**→**get\_functionId()**

**YDataLogger**

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

---

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

**datalogger→get\_hardwareId()****YDataLogger****datalogger→hardwareId()****datalogger.get\_hardwareId()**

---

Retourne l'identifiant matériel unique de l'enregistreur de données au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'enregistreur de données (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**datalogger→get\_logicalName()**

**YDataLogger**

**datalogger→logicalName()**

**datalogger.get\_logicalName()**

---

Retourne le nom logique de l'enregistreur de données.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'enregistreur de données. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.



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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**datalogger**→**get\_recording()**

**YDataLogger**

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

---

Retourne l'état d'activation de l'enregistreur de données.

```
def get_recording( )
```

**Retourne :**

soit Y\_RECORDING\_OFF, soit Y\_RECORDING\_ON, selon l'état d'activation de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_RECORDING\_INVALID.

**datalogger→get\_timeUTC()****YDataLogger****datalogger→timeUTC()datalogger.get\_timeUTC()**

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

```
def get_timeUTC( )
```

**Retourne :**

un entier représentant le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue

En cas d'erreur, déclenche une exception ou retourne Y\_TIMEUTC\_INVALID.

**datalogger**→**get\_userData()**

**YDataLogger**

**datalogger**→**userData()****datalogger.get\_userData()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

**datalogger**→**isOnline()****datalogger.isOnline()****YDataLogger**

---

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'enregistreur de données est joignable, `false` sinon

**datalogger**→**load()****datalogger.load()****YDataLogger**

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**datalogger**→**nextDataLogger()**  
**datalogger.nextDataLogger()**

---

**YDataLogger**

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

```
def nextDataLogger( )
```

**Retourne :**

un pointeur sur un objet `YDataLogger` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**datalogger**→**registerValueCallback()**  
**datalogger.registerValueCallback()****YDataLogger**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.



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

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

```
def set_autoStart( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→set\_logicalName()****YDataLogger****datalogger→setLogicalName()****datalogger.set\_logicalName()**

Modifie le nom logique de l'enregistreur de données.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'enregistreur de données.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Modifie l'état d'activation de l'enregistreur de données.

```
def set_recording( newval)
```

**Paramètres :**

**newval** soit Y\_RECORDING\_OFF, soit Y\_RECORDING\_ON, selon l'état d'activation de l'enregistreur de données

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→set\_timeUTC()**

**YDataLogger**

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

---

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

```
def set_timeUTC( newval)
```

**Paramètres :**

**newval** un entier représentant la référence de temps UTC, afin de l'attacher aux données enregistrées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger**→**set\_userdata()****YDataLogger****datalogger**→**setUserData()****datalogger.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.9. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Méthodes des objets YDataRun

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

Retourne la valeur moyenne des mesures observées au moment choisi.

#### **datarun→get\_duration()**

Retourne la durée (en secondes) du Run.

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

Retourne la valeur maximale des mesures observées au moment choisi.

#### **datarun→get\_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

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

Retourne la valeur minimale des mesures observées au moment choisi.

#### **datarun→get\_startTimeUTC()**

Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### **datarun→get\_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

#### **datarun→get\_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

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

Change l'intervalle de temps représenté par chaque valeur de ce run.

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

Retourne la valeur moyenne des mesures observées au moment choisi.

```
def get_averageValue( measureName, pos)
```

**Paramètres :**

**measureName** le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

**pos** l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

**Retourne :**

une nombre flottant (la valeur moyenne).

En cas d'erreur, déclenche une exception ou retourne `Y_AVERAGEVALUE_INVALID`.

**datarun**→**get\_duration()**

**YDataRun**

**datarun**→**duration()****datarun.get\_duration()**

---

Retourne la durée (en secondes) du Run.

**def get\_duration( )**

Lorsque cette méthode est appelée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le début du Run (quand le module a été mis sous tension) et la dernière mesure enregistrée.



**datarun**→**get\_maxValue()****YDataRun****datarun**→**maxValue()****datarun.get\_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

```
def get_maxValue( measureName, pos)
```

**Paramètres :**

**measureName** le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

**pos** l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

**Retourne :**

une nombre flottant (la valeur maximale).

En cas d'erreur, déclenche une exception ou retourne `Y_MAXVALUE_INVALID`.

**datarun→get\_measureNames()**

**YDataRun**

**datarun→measureNames()**

**datarun.get\_measureNames()**

---

Retourne les noms des valeurs mesurées par l'enregistreur de données.

def **get\_measureNames**( )

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure.

**Retourne :**

une liste de chaîne de caractères (les noms des mesures)

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**datarun**→**get\_minValue()****YDataRun****datarun**→**minValue()****datarun.get\_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

def **get\_minValue**( **measureName**, **pos**)

**Paramètres :**

**measureName** le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

**pos** l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

**Retourne :**

une nombre flottant (la valeur minimale).

En cas d'erreur, déclenche une exception ou retourne `Y_MINVALUE_INVALID`.

**datarun→get\_startTimeUTC()**

**YDataRun**

**datarun→startTimeUTC()**

---

Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

Si l'heure UTC n'a jamais été configurée dans l'enregistreur de données durant le run, et si il ne s'agit pas du run courant, cette méthode retourne 0.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début du Run.

**datarun→get\_valueCount()****YDataRun****datarun→valueCount()****datarun.get\_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

**def get\_valueCount( )**

Lorsque cette méthode est appelée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

**Retourne :**

un entier positif correspondant à la durée du Run divisée par l'intervalle entre les valeurs.

**datarun**→**get\_valueInterval()**

**YDataRun**

**datarun**→**valueInterval()****datarun.get\_valueInterval()**

---

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

def **get\_valueInterval()** ( )

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

**Retourne :**

un entier positif correspondant au nombre de secondes couvertes par chaque valeur représentée dans le Run.

**datarun**→**set\_valueInterval()**  
**datarun**→**setValueInterval()**  
**datarun.set\_valueInterval()**

**YDataRun**

---

Change l'intervalle de temps représenté par chaque valeur de ce run.

```
def set_valueInterval( valueInterval)
```

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

**Paramètres :**

**valueInterval** un nombre entier de secondes.

**Retourne :**

nothing

## 3.10. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-même sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

Cette classe ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Méthodes des objets YDataSet

#### **dataset→get\_endTimeUTC()**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### **dataset→get\_functionId()**

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

#### **dataset→get\_hardwareId()**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

#### **dataset→get\_measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

#### **dataset→get\_preview()**

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

#### **dataset→get\_progress()**

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

#### **dataset→get\_startTimeUTC()**

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### **dataset→get\_summary()**

Retourne un objet YMeasure résumant tout le YDataSet.

#### **dataset→get\_unit()**



Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**dataset→loadMore()**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

**dataset→loadMore\_async(callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**dataset**→**get\_endTimeUTC()**

**YDataSet**

**dataset**→**endTimeUTC()****dataset.get\_endTimeUTC()**

---

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

def **get\_endTimeUTC()** ( )

Lorsque l'objet YDataSet est créé, l'heure de fin est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de fin est mise à jour à la dernière mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la dernière mesure.

---

**dataset**→**get\_functionId()****YDataSet****dataset**→**functionId()****dataset.get\_functionId()**

---

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

```
def get_functionId( )
```

Par exemple `temperature1`.

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: `temperature1`)

**dataset**→**get\_hardwareId()**

**YDataSet**

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

---

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple `THRMCPL1-123456.temperature1`).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: `THRMCPL1-123456.temperature1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**dataset**→**get\_measures()****YDataSet****dataset**→**measures()****dataset.get\_measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

```
def get_measures( )
```

Chaque élément contient: - le moment où la mesure a débuté - le moment où la mesure s'est terminée - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Avant d'appeler cette méthode, vous devez appeler `loadMore()` pour charger des données depuis l'enregistreur sur le module. L'appel doit être répété plusieurs fois pour charger toutes les données, mais vous pouvez commencer à utiliser les données disponibles avant qu'elles n'aient été toutes chargées

Les mesures les plus anciennes sont toujours chargées les premières, et les plus récentes en dernier. De ce fait, les timestamps dans la table des mesures sont normalement par ordre chronologique. La seule exception est dans le cas où il y a eu un ajustement de l'horloge UTC de l'enregistreur de données pendant l'enregistrement.

**Retourne :**

un tableau d'enregistrements, chaque enregistrement représentant une mesure effectuée à un moment précis.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**dataset**→**get\_preview()**

**YDataSet**

**dataset**→**preview()****dataset.get\_preview()**

---

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

```
def get_preview( )
```

Chaque élément contient: - le début d'un intervalle de temps - la fin d'un intervalle de temps - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Le résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

**Retourne :**

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**dataset**→**get\_progress()****YDataSet****dataset**→**progress()****dataset.get\_progress()**

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

```
def get_progress( )
```

A l'instanciation de l'objet par la fonction `get_dataSet( )`, l'avancement est nul. Au fur et à mesure des appels à `loadMore( )`, l'avancement progresse pour atteindre la valeur 100 lorsque toutes les mesures ont été chargées.

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées.

**dataset**→**get\_startTimeUTC()**

**YDataSet**

**dataset**→**startTimeUTC()****dataset.get\_startTimeUTC()**

---

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

def **get\_startTimeUTC()**

Lorsque l'objet YDataSet est créé, l'heure de départ est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de départ est mise à jour à la première mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la première mesure enregistrée.



**dataset**→**get\_summary()****YDataSet****dataset**→**summary()****dataset.get\_summary()**

Retourne un objet YMeasure résumant tout le YDataSet.

```
def get_summary( )
```

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `loadMore( )` a été appelé pour la première fois.

**Retourne :**

un objet YMeasure

**dataset**→**get\_unit()**

**YDataSet**

**dataset**→**unit()****dataset.get\_unit()**

---

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant une unité physique.

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

---

**dataset**→**loadMore()****dataset.loadMore()****YDataSet**

---

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

def **loadMore**( )

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.11. Séquence de données enregistrées brute

Les objets YDataStream correspondent aux séquences de mesures enregistrées brutes, directement telles qu'obtenues par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Dans la plupart des cas, il n'est pas nécessaire d'utiliser les objets DataStream, car les objets YDataSet (retournés par la méthode `get_recordedData()` des senseurs et la méthode `get_dataSets()` du DataLogger) fournissent une interface plus pratique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Méthodes des objets YDataStream

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

Retourne la moyenne des valeurs observées durant cette séquence.

#### **datastream→get\_columnCount()**

Retourne le nombre de colonnes de données contenus dans la séquence.

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

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

#### **datastream→get\_data(row, col)**

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

#### **datastream→get\_dataRows()**

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

#### **datastream→get\_dataSamplesIntervalMs()**

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

#### **datastream→get\_duration()**

Retourne la durée approximative de cette séquence, en secondes.

#### **datastream→get\_maxValue()**

Retourne la plus grande valeur observée durant cette séquence.

#### **datastream→get\_minValue()**

Retourne la plus petite valeur observée durant cette séquence.

#### **datastream→get\_rowCount()**

Retourne le nombre d'enregistrement contenus dans la séquence.

#### **datastream→get\_runIndex()**

Retourne le numéro de Run de la séquence de données.

#### **datastream→get\_startTime()**

Retourne le temps de départ relatif de la séquence (en secondes).

**`datastream→get_startTimeUTC()`**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

**datastream→get\_averageValue()**

**YDataStream**

**datastream→averageValue()**

**datastream.get\_averageValue()**

---

Retourne la moyenne des valeurs observées durant cette séquence.

```
def get_averageValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la moyenne des valeurs, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→get\_columnCount()****YDataStream****datastream→columnCount()****datastream.get\_columnCount()**

---

Retourne le nombre de colonnes de données contenus dans la séquence.

```
def get_columnCount( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames( )`.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre de colonnes.

En cas d'erreur, déclenche une exception ou retourne zéro.

**datastream→get\_columnNames()**

**YDataStream**

**datastream→columnNames()**

**datastream.get\_columnNames()**

---

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

```
def get_columnNames( )
```

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure. Pour les séquences enregistrées à faible fréquence, l'enregistreur de donnée stocke la valeur min, moyenne et max observée durant chaque intervalle de temps dans des colonnes avec les suffixes `_min`, `_avg` et `_max` respectivement.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

une liste de chaîne de caractères.

En cas d'erreur, déclenche une exception ou retourne une liste vide.



**datastream**→**get\_data()****YDataStream****datastream**→**data()****datastream.get\_data()**

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

```
def get_data( row, col)
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Paramètres :**

**row** index de l'enregistrement (ligne)

**col** index de la mesure (colonne)

**Retourne :**

un nombre décimal

En cas d'erreur, déclenche une exception ou retourne `Y_DATA_INVALID`.

**datastream**→**get\_dataRows()**

**YDataStream**

**datastream**→**dataRows()****datastream.get\_dataRows()**

---

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

```
def get_dataRows( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Retourne :**

une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

---

**datastream→get\_dataSamplesIntervalMs()****YDataStream****datastream→dataSamplesIntervalMs()****datastream.get\_dataSamplesIntervalMs()**

---

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

```
def get_dataSamplesIntervalMs( )
```

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la fréquence d'enregistrement peut être changée pour chaque fonction.

**Retourne :**

un entier positif correspondant au nombre de millisecondes entre deux mesures consécutives.

**datastream**→**get\_duration()**

**YDataStream**

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

---

Retourne la durée approximative de cette séquence, en secondes.

```
def get_duration( )
```

**Retourne :**

le nombre de secondes couvertes par cette séquence.

En cas d'erreur, déclenche une exception ou retourne Y\_DURATION\_INVALID.

---

**datastream**→**get\_maxValue()****YDataStream****datastream**→**maxValue()****datastream.get\_maxValue()**

---

Retourne la plus grande valeur observée durant cette séquence.

```
def get_maxValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus grande valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream**→**get\_minValue()**

**YDataStream**

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

---

Retourne la plus petite valeur observée durant cette séquence.

```
def get_minValue()
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus petite valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream**→**get\_rowCount()****YDataStream****datastream**→**rowCount()****datastream.getRowCount()**

Retourne le nombre d'enregistrement contenus dans la séquence.

```
def get_rowCount( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre d'enregistrements.

En cas d'erreur, déclenche une exception ou retourne zéro.

**datastream**→**get\_runIndex()**

**YDataStream**

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

---

Retourne le numéro de Run de la séquence de données.

```
def get_runIndex( )
```

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

**Retourne :**

un entier positif correspondant au numéro du Run



---

**datastream**→**get\_startTime()****YDataStream****datastream**→**startTime()****datastream.get\_startTime()**

---

Retourne le temps de départ relatif de la séquence (en secondes).

```
def get_startTime( )
```

Pour les firmwares récents, la valeur est relative à l'heure courante (valeur négative). Pour les modules utilisant un firmware plus ancien que la version 13000, la valeur est le nombre de secondes depuis la mise sous tension du module (valeur positive). Si vous désirez obtenir l'heure absolue du début de la séquence, utilisez `get_startTimeUTC( )`.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le début du Run et le début de la séquence enregistrée.

**datastream→get\_startTimeUTC()**

**YDataStream**

**datastream→startTimeUTC()**

**datastream.get\_startTimeUTC()**

---

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

```
def get_startTimeUTC( )
```

Si l'heure UTC n'était pas configurée dans l'enregistreur de données au début de la séquence, cette méthode retourne 0.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début de la séquence enregistrée.

## 3.12. Interface de la fonction DigitalIO

La librairie de programmation Yoctopuce permet simplement de changer l'état de chaque bit du port d'entrée sortie. Il est possible de changer tous les bits du port à la fois, ou de les changer indépendamment. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Le comportement électrique de chaque entrée/sortie peut être modifié (open drain et polarité inverse).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindDigitalIO(func)

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

#### yFirstDigitalIO()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

### Méthodes des objets YDigitalIO

#### digitalio→delayedPulse(bitno, ms\_delay, ms\_duration)

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

#### digitalio→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### digitalio→get\_advertisedValue()

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

#### digitalio→get\_bitDirection(bitno)

Retourne la direction d'un seul bit du port d'E/S.

#### digitalio→get\_bitOpenDrain(bitno)

Retourne la direction d'un seul bit du port d'E/S.

#### digitalio→get\_bitPolarity(bitno)

Retourne la polarité d'un seul bit du port d'E/S.

#### digitalio→get\_bitState(bitno)

Retourne l'état d'un seul bit du port d'E/S.

#### digitalio→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

#### digitalio→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

#### digitalio→get\_friendlyName()

Retourne un identifiant global du port d'E/S digital au format NOM\_MODULE . NOM\_FONCTION.

#### digitalio→get\_functionDescriptor()

	Retourne un identifiant unique de type <code>YFUN_DESCR</code> correspondant à la fonction.
<b>digitalio→get_functionId()</b>	Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.
<b>digitalio→get_hardwareId()</b>	Retourne l'identifiant matériel unique du port d'E/S digital au format <code>SERIAL . FUNCTIONID</code> .
<b>digitalio→get_logicalName()</b>	Retourne le nom logique du port d'E/S digital.
<b>digitalio→get_module()</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>digitalio→get_module_async(callback, context)</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>digitalio→get_outputVoltage()</b>	Retourne la source de tension utilisée pour piloter les bits en sortie.
<b>digitalio→get_portDirection()</b>	Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.
<b>digitalio→get_portOpenDrain()</b>	Retourne le type d'interface électrique de chaque bit du port (bitmap).
<b>digitalio→get_portPolarity()</b>	Retourne la polarité des bits du port (bitmap).
<b>digitalio→get_portSize()</b>	Retourne le nombre de bits implémentés dans le port d'E/S.
<b>digitalio→get_portState()</b>	Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.
<b>digitalio→get_userData()</b>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<b>digitalio→isOnline()</b>	Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.
<b>digitalio→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.
<b>digitalio→load(msValidity)</b>	Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.
<b>digitalio→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.
<b>digitalio→nextDigitalIO()</b>	Continue l'énumération des ports d'E/S digitaux commencée à l'aide de <code>yFirstDigitalIO()</code> .
<b>digitalio→pulse(bitno, ms_duration)</b>	Déclenche une impulsion de durée spécifiée sur un bit choisi.
<b>digitalio→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>digitalio→set_bitDirection(bitno, bitdirection)</b>	Change la direction d'un seul bit du port d'E/S.
<b>digitalio→set_bitOpenDrain(bitno, opendrain)</b>	Change le type d'interface électrique d'un seul bit du port d'E/S.
<b>digitalio→set_bitPolarity(bitno, bitpolarity)</b>	Change la polarité d'un seul bit du port d'E/S.

**digitalio→set\_bitState(bitno, bitstate)**

Change l'état d'un seul bit du port d'E/S.

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

Modifie le nom logique du port d'E/S digital.

**digitalio→set\_outputVoltage(newval)**

Modifie la source de tension utilisée pour piloter les bits en sortie.

**digitalio→set\_portDirection(newval)**

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

**digitalio→set\_portOpenDrain(newval)**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

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

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne de manière inversée.

**digitalio→set\_portState(newval)**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

**digitalio→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**digitalio→toggle\_bitState(bitno)**

Inverse l'état d'un seul bit du port d'E/S.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YDigitalIO.FindDigitalIO() yFindDigitalIO()YDigitalIO.FindDigitalIO()

YDigitalIO

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

```
def FindDigitalIO( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port d'E/S digital soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDigitalIO.isOnline()` pour tester si le port d'E/S digital est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le port d'E/S digital sans ambiguïté

### Retourne :

un objet de classe `YDigitalIO` qui permet ensuite de contrôler le port d'E/S digital.

---

**YDigitalIO.FirstDigitalIO()**  
**yFirstDigitalIO()YDigitalIO.FirstDigitalIO()**

---

**YDigitalIO**

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

```
def FirstDigitalIO( )
```

Utiliser la fonction `YDigitalIO.nextDigitalIO( )` pour itérer sur les autres ports d'E/S digitaux.

**Retourne :**

un pointeur sur un objet `YDigitalIO`, correspondant au premier port d'E/S digital accessible en ligne, ou `null` si il n'y a pas de ports d'E/S digitaux disponibles.

**digitalio→delayedPulse()digitalio.delayedPulse()****YDigitalIO**

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

```
def delayedPulse( bitno, ms_delay, ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

- bitno** index du bit dans le port; le bit de poids faible est à l'index 0
- ms\_delay** délai d'attente avant l'impulsion, en millisecondes
- ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**digitalio→describe()digitalio.describe()****YDigitalIO**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le port d'E/S digital (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

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

**YDigitalIO**

**digitalio**→**advertisedValue()**

**digitalio.get\_advertisedValue()**

---

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du port d'E/S digital (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**digitalio**→**get\_bitDirection()****YDigitalIO****digitalio**→**bitDirection()****digitalio.get\_bitDirection()**

Retourne la direction d'un seul bit du port d'E/S.

```
def get_bitDirection( bitno)
```

(0 signifie que le bit est une entrée, 1 une sortie)

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**get\_bitOpenDrain()**

**YDigitalIO**

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

Retourne la direction d'un seul bit du port d'E/S.

```
def get_bitOpenDrain( bitno)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert)..

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**get\_bitPolarity()****YDigitalIO****digitalio**→**bitPolarity()****digitalio.get\_bitPolarity()**

Retourne la polarité d'un seul bit du port d'E/S.

```
def get_bitPolarity( bitno)
```

0 signifie que l'entrée sortie est en mode normal, 1 qu'elle est en mode inverse

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**get\_bitState()**

**YDigitalIO**

**digitalio**→**bitState()****digitalio.get\_bitState()**

---

Retourne l'état d'un seul bit du port d'E/S.

```
def get_bitState( bitno)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

l'état du bit (0 ou 1).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**digitalio**→**get\_errorMessage()****YDigitalIO****digitalio**→**errorMessage()****digitalio.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

**digitalio**→**get\_errorType()**

**YDigitalIO**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.



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

Retourne un identifiant global du port d'E/S digital au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du port d'E/S digital si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port d'E/S digital (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YDigitalIO**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

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

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**digitalio**→**get\_hardwareId()**

**YDigitalIO**

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

---

Retourne l'identifiant matériel unique du port d'E/S digital au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port d'E/S digital (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

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

Retourne le nom logique du port d'E/S digital.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port d'E/S digital. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**digitalio**→**get\_module()**

**YDigitalIO**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

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

**YDigitalIO**

Retourne la source de tension utilisée pour piloter les bits en sortie.

```
def get_outputVoltage( )
```

**Retourne :**

une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUTVOLTAGE\_INVALID.

**digitalio**→**get\_portDirection()**

**YDigitalIO**

**digitalio**→**portDirection()****digitalio.get\_portDirection()**

---

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
def get_portDirection( )
```

**Retourne :**

un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

En cas d'erreur, déclenche une exception ou retourne Y\_PORTDIRECTION\_INVALID.



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

**YDigitalIO**

---

Retourne le type d'interface électrique de chaque bit du port (bitmap).

```
def get_portOpenDrain( )
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

**Retourne :**

un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y\_PORTOPENDRAIN\_INVALID.

**digitalio**→**get\_portPolarity()**

**YDigitalIO**

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

---

Retourne la polarité des bits du port (bitmap).

```
def get_portPolarity( )
```

Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

**Retourne :**

un entier représentant la polarité des bits du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y\_PORTPOLARITY\_INVALID.

**digitalio**→**get\_portSize()****YDigitalIO****digitalio**→**portSize()****digitalio.get\_portSize()**

Retourne le nombre de bits implémentés dans le port d'E/S.

```
def get_portSize( )
```

**Retourne :**

un entier représentant le nombre de bits implémentés dans le port d'E/S

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSIZE_INVALID`.

**digitalio**→**get\_portState()**

**YDigitalIO**

**digitalio**→**portState()****digitalio.get\_portState()**

---

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
def get_portState( )
```

**Retourne :**

un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

En cas d'erreur, déclenche une exception ou retourne Y\_PORTSTATE\_INVALID.

---

**digitalio**→**get\_userData()****YDigitalIO****digitalio**→**userData()****digitalio.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le port d'E/S digital est joignable, `false` sinon

**digitalio**→**load()****digitalio.load()****YDigitalIO**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**nextDigitalIO()****digitalio.nextDigitalIO()**

**YDigitalIO**

---

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

def **nextDigitalIO**( )

**Retourne :**

un pointeur sur un objet `YDigitalIO` accessible en ligne, ou `null` lorsque l'énumération est terminée.



**digitalio**→**pulse()****digitalio.pulse()****YDigitalIO**

Déclenche une impulsion de durée spécifiée sur un bit choisi.

```
def pulse( bitno, ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0  
**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→registerValueCallback()**  
**digitalio.registerValueCallback()****YDigitalIO**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**digitalio**→**set\_bitDirection()****YDigitalIO****digitalio**→**setBitDirection()****digitalio.set\_bitDirection()**

Change la direction d'un seul bit du port d'E/S.

```
def set_bitDirection( bitno, bitdirection)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitdirection** nouvelle valeur de la direction, 0=entrée, 1=sortie. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_bitOpenDrain()**  
**digitalio**→**setBitOpenDrain()**  
**digitalio.set\_bitOpenDrain()**

**YDigitalIO**

Change le type d'interface électrique d'un seul bit du port d'E/S.

```
def set_bitOpenDrain( bitno, opendrain)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0  
**opendrain** 0 pour faire une entrée ou une sortie digitale standard, 1 pour une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_bitPolarity()****YDigitalIO****digitalio**→**setBitPolarity()****digitalio.set\_bitPolarity()**

Change la polarité d'un seul bit du port d'E/S.

```
def set_bitPolarity( bitno, bitpolarity)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitpolarity** nouvelle valeur de la polarité. 0=mode normal, 1=mode inverse. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_bitState()****YDigitalIO****digitalio**→**setBitState()****digitalio.set\_bitState()**

Change l'état d'un seul bit du port d'E/S.

```
def set_bitState( bitno, bitstate)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitstate** nouvel état du bit (1 ou 0)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_logicalName()****YDigitalIO****digitalio**→**setLogicalName()****digitalio.set\_logicalName()**

Modifie le nom logique du port d'E/S digital.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du port d'E/S digital.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_outputVoltage()****YDigitalIO****digitalio**→**setOutputVoltage()****digitalio.set\_outputVoltage()**

Modifie la source de tension utilisée pour piloter les bits en sortie.

```
def set_outputVoltage( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Paramètres :**

**newval** une valeur parmi `Y_OUTPUTVOLTAGE_USB_5V`, `Y_OUTPUTVOLTAGE_USB_3V` et `Y_OUTPUTVOLTAGE_EXT_V` représentant la source de tension utilisée pour piloter les bits en sortie

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**digitalio**→**set\_portDirection()**  
**digitalio**→**setPortDirection()**  
**digitalio.set\_portDirection()**

**YDigitalIO**

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
def set_portDirection( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_portOpenDrain()**  
**digitalio**→**setPortOpenDrain()**  
**digitalio.set\_portOpenDrain()**

**YDigitalIO**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

```
def set_portOpenDrain( newval)
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

```
def set_portPolarity( newval)
```

**Paramètres :**

**newval** un entier représentant la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_portState()****YDigitalIO****digitalio**→**setPortState()****digitalio.set\_portState()**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
def set_portState( newval)
```

Seuls les bits configurés en sortie dans `portDirection` sont affectés.

**Paramètres :**

**newval** un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio**→**set\_userdata()****YDigitalIO****digitalio**→**setUserData()****digitalio.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**digitalio**→**toggle\_bitState()****digitalio.toggle\_bitState()****YDigitalIO**

Inverse l'état d'un seul bit du port d'E/S.

```
def toggle_bitState( bitno)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

### 3.13. Interface de la fonction Display

L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

#### Fonction globales

##### yFindDisplay(func)

Permet de retrouver un écran d'après un identifiant donné.

##### yFirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

#### Méthodes des objets YDisplay

##### display→copyLayerContent(srcLayerId, dstLayerId)

Copie le contenu d'un couche d'affichage vers une autre couche.

##### display→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format TYPE (NAME) = SERIAL . FUNCTIONID.

##### display→fade(brightness, duration)

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

##### display→get\_advertisedValue()

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

##### display→get\_brightness()

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

##### display→get\_displayHeight()

Retourne la hauteur de l'écran, en pixels.

##### display→get\_displayLayer(layerId)

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

##### display→get\_displayType()

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

##### display→get\_displayWidth()

Retourne la largeur de l'écran, en pixels.

##### display→get\_enabled()

Retourne vrai si le l'écran est alimenté, faux sinon.

##### display→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

#### **display→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

#### **display→get\_friendlyName()**

Retourne un identifiant global de l'écran au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **display→get\_functionId()**

Retourne l'identifiant matériel de l'écran, sans référence au module.

#### **display→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'écran au format `SERIAL . FUNCTIONID`.

#### **display→get\_layerCount()**

Retourne le nombre des couches affichables disponibles.

#### **display→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

#### **display→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

#### **display→get\_logicalName()**

Retourne le nom logique de l'écran.

#### **display→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **display→get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **display→get\_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

#### **display→get\_startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

#### **display→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **display→isOnline()**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

#### **display→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

#### **display→load(msValidity)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

#### **display→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

#### **display→newSequence()**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

#### **display→nextDisplay()**

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

#### **display→pauseSequence(delay\_ms)**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.



**display→playSequence(sequenceName)**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

**display→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**display→resetAll()**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

**display→saveSequence(sequenceName)**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

**display→set\_brightness(newval)**

Modifie la luminosité de l'écran.

**display→set\_enabled(newval)**

Modifie l'état d'activité de l'écran.

**display→set\_logicalName(newval)**

Modifie le nom logique de l'écran.

**display→set\_orientation(newval)**

Modifie l'orientation de l'écran.

**display→set\_startupSeq(newval)**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

**display→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**display→stopSequence(sequenceName)**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

**display→swapLayerContent(layerIdA, layerIdB)**

Permute le contenu de deux couches d'affichage.

**display→upload(pathname, content)**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YDisplay.FindDisplay() yFindDisplay()YDisplay.FindDisplay()

YDisplay

Permet de retrouver un ecran d'après un identifiant donné.

```
def FindDisplay( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'ecran soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDisplay.isOnline()` pour tester si l'ecran est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'ecran sans ambiguïté

### Retourne :

un objet de classe `YDisplay` qui permet ensuite de contrôler l'ecran.

## **YDisplay.FirstDisplay()** **yFirstDisplay()YDisplay.FirstDisplay()**

## **YDisplay**

Commence l'énumération des écran accessibles par la librairie.

```
def FirstDisplay( )
```

Utiliser la fonction `YDisplay.nextDisplay( )` pour itérer sur les autres écran.

### **Retourne :**

un pointeur sur un objet `YDisplay`, correspondant au premier écran accessible en ligne, ou `null` si il n'y a pas de écran disponibles.

**display→copyLayerContent()**  
**display.copyLayerContent()****YDisplay**

Copie le contenu d'une couche d'affichage vers une autre couche.

```
def copyLayerContent( srcLayerId, dstLayerId)
```

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**srcLayerId** l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

**dstLayerId** l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→describe()display.describe()****YDisplay**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'ecran au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'ecran (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

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

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

```
def fade( brightness, duration)
```

**Paramètres :**

**brightness** nouvelle valeur de luminosité de l'écran

**duration** durée en millisecondes de la transition.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

Retourne la valeur courante de l'ecran (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'ecran (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**display**→**get\_brightness()**

**YDisplay**

**display**→**brightness()****display.get\_brightness()**

---

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
def get_brightness( )
```

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne `Y_BRIGHTNESS_INVALID`.



**display**→**get\_displayHeight()****YDisplay****display**→**displayHeight()****display.get\_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

```
def get_displayHeight( )
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYHEIGHT\_INVALID.

**display**→**get\_displayLayer()**

**YDisplay**

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

---

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

```
def get_displayLayer( layerId)
```

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

**Paramètres :**

**layerId** l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

**Retourne :**

un objet YDisplayLayer

En cas d'erreur, déclenche une exception ou retourne `null`.

**display**→**get\_displayType()****YDisplay****display**→**displayType()****display.get\_displayType()**

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

```
def get_displayType( )
```

**Retourne :**

une valeur parmi Y\_DISPLAYTYPE\_MONO, Y\_DISPLAYTYPE\_GRAY et Y\_DISPLAYTYPE\_RGB représentant le type de l'écran: monochrome, niveaux de gris ou couleur

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYTYPE\_INVALID.

**display**→**get\_displayWidth()**

**YDisplay**

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

---

Retourne la largeur de l'écran, en pixels.

```
def get_displayWidth( )
```

**Retourne :**

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_DISPLAYWIDTH_INVALID`.

**display**→**get\_enabled()****YDisplay****display**→**enabled()****display.get\_enabled()**

Retourne vrai si le l'ecran est alimenté, faux sinon.

```
def get_enabled( )
```

**Retourne :**

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon vrai si le l'ecran est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

**display**→**get\_errorMessage()**

**YDisplay**

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

**display**→**get\_errorType()****YDisplay****display**→**errorType()****display.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

**display**→**get\_friendlyName()**

**YDisplay**

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

---

Retourne un identifiant global de l'ecran au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'ecran si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'ecran (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'ecran en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.



---

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

---

**YDisplay**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

**display**→**get\_functionId()**

**YDisplay**

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

---

Retourne l'identifiant matériel de l'ecran, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'ecran (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**display**→**get\_hardwareId()****YDisplay****display**→**hardwareId()****display.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'écran au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'écran (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'écran (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**display**→**get\_layerCount()**

**YDisplay**

**display**→**layerCount()****display.get\_layerCount()**

---

Retourne le nombre des couches affichables disponibles.

```
def get_layerCount( )
```

**Retourne :**

un entier représentant le nombre des couches affichables disponibles

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERCOUNT_INVALID`.

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

Retourne la hauteur des couches affichables, en pixels.

```
def get_layerHeight( )
```

**Retourne :**

un entier représentant la hauteur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERHEIGHT_INVALID`.

**display**→**get\_layerWidth()**

**YDisplay**

**display**→**layerWidth()****display.get\_layerWidth()**

---

Retourne la largeur des couches affichables, en pixels.

```
def get_layerWidth( )
```

**Retourne :**

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERWIDTH_INVALID`.

**display**→**get\_logicalName()****YDisplay****display**→**logicalName()****display.get\_logicalName()**

Retourne le nom logique de l'ecran.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'ecran. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**display→get\_module()**

**YDisplay**

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

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule



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

Retourne l'orientation sélectionnée pour l'écran.

```
def get_orientation( )
```

**Retourne :**

une valeur parmi Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP, Y\_ORIENTATION\_RIGHT et Y\_ORIENTATION\_DOWN représentant l'orientation sélectionnée pour l'écran

En cas d'erreur, déclenche une exception ou retourne Y\_ORIENTATION\_INVALID.

**display**→**get\_startupSeq()**

**YDisplay**

**display**→**startupSeq()****display.get\_startupSeq()**

---

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

```
def get_startupSeq( )
```

**Retourne :**

une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

En cas d'erreur, déclenche une exception ou retourne Y\_STARTUPSEQ\_INVALID.

**display**→**get\_userData()****YDisplay****display**→**userData()****display.get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

**def get\_userData( )**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **display**→**isOnline()****display.isOnline()**

**YDisplay**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'écran sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'écran est joignable, `false` sinon

**display→load()display.load()****YDisplay**

Met en cache les valeurs courantes de l'ecran, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **display**→**newSequence()****display.newSequence()**

**YDisplay**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

```
def newSequence( )
```

Le nom de la séquence sera donné au moment de l'appel à `saveSequence( )`, une fois la séquence terminée.

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**display**→**nextDisplay()****display.nextDisplay()****YDisplay**

---

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

```
def nextDisplay( )
```

**Retourne :**

un pointeur sur un objet `YDisplay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**display→pauseSequence()display.pauseSequence()****YDisplay**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

```
def pauseSequence( delay_ms)
```

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

**Paramètres :**

**delay\_ms** la durée de l'attente, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**display→playSequence()display.playSequence()****YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

```
def playSequence( sequenceName)
```

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→registerValueCallback()**  
**display.registerValueCallback()****YDisplay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

**display→resetAll()display.resetAll()****YDisplay**

---

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

```
def resetAll( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→saveSequence()display.saveSequence()****YDisplay**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

```
def saveSequence( sequenceName)
```

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Modifie la luminosité de l'écran.

```
def set_brightness( newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la luminosité de l'écran

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display**→**set\_enabled()**  
**display**→**setEnabled()****display.set\_enabled()**

**YDisplay**

Modifie l'état d'activité de l'écran.

```
def set_enabled( newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état d'activité de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

Modifie le nom logique de l'ecran.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'ecran.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display**→**set\_orientation()**

**YDisplay**

**display**→**setOrientation()****display.set\_orientation()**

---

Modifie l'orientation de l'écran.

```
def set_orientation( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi `Y_ORIENTATION_LEFT`, `Y_ORIENTATION_UP`, `Y_ORIENTATION_RIGHT` et `Y_ORIENTATION_DOWN` représentant l'orientation de l'écran

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**display**→**set\_startupSeq()****YDisplay****display**→**setStartupSeq()****display.set\_startupSeq()**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
def set_startupSeq( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display**→**set\_userdata()**

**YDisplay**

**display**→**setUserData()****display.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

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

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

```
def stopSequence( )
```

L'affichage est laissé tel quel.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→swapLayerContent()**  
**display.swapLayerContent()****YDisplay**

Permute le contenu de deux couches d'affichage.

```
def swapLayerContent( layerIdA, layerIdB)
```

La couleur et la transparence de tous les pixels des deux couches sont permutées. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**layerIdA** l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

**layerIdB** l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→upload()display.upload()****YDisplay**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

```
def upload( pathname, content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.14. Interface des objets DisplayLayer

Un DisplayLayer est une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Méthodes des objets YDisplayLayer

#### displaylayer→clear()

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

#### displaylayer→clearConsole(text)

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

#### displaylayer→consoleOut(text)

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

#### displaylayer→drawBar(x1, y1, x2, y2)

Dessine un rectangle plein à une position spécifiée.

#### displaylayer→drawBitmap(x, y, w, bitmap, bgcol)

Dessine un bitmap à la position spécifiée de la couche.

#### displaylayer→drawCircle(x, y, r)

Dessine un cercle vide à une position spécifiée.

#### displaylayer→drawDisc(x, y, r)

Dessine un disque plein à une position spécifiée.

#### displaylayer→drawImage(x, y, imagename)

Dessine une image GIF à la position spécifiée de la couche.

#### displaylayer→drawPixel(x, y)

Dessine un pixel unique à une position spécifiée.

#### displaylayer→drawRect(x1, y1, x2, y2)

Dessine un rectangle vide à une position spécifiée.

#### displaylayer→drawText(x, y, anchor, text)

Affiche un texte à la position spécifiée de la couche.

#### displaylayer→get\_display()

Retourne l'YDisplay parent.

#### displaylayer→get\_displayHeight()

Retourne la hauteur de l'écran, en pixels.

#### displaylayer→get\_displayWidth()

Retourne la largeur de l'écran, en pixels.

**displaylayer→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

**displaylayer→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

**displaylayer→hide()**

Cache la couche de dessin.

**displaylayer→lineTo(x, y)**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

**displaylayer→moveTo(x, y)**

Déplace le point de dessin courant de cette couche à la position spécifiée.

**displaylayer→reset()**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

**displaylayer→selectColorPen(color)**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

**displaylayer→selectEraser()**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

**displaylayer→selectFont(fontname)**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

**displaylayer→selectGrayPen(graylevel)**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

**displaylayer→setAntialiasingMode(mode)**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

**displaylayer→setConsoleBackground(bgcol)**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

**displaylayer→setConsoleMargins(x1, y1, x2, y2)**

Configure les marges d'affichage pour la fonction `consoleOut`.

**displaylayer→setConsoleWordWrap(wordwrap)**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

**displaylayer→setLayerPosition(x, y, scrollTime)**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

**displaylayer→unhide()**

Affiche la couche.

## **displaylayer→clear()displaylayer.clear()**

**YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

```
def clear( )
```

Cette méthode ne change pas les réglages de la couche. Si vous désirez remettre la couche dans son état initial, utilisez plutôt la méthode `reset()`.

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**displaylayer→clearConsole()**  
**displaylayer.clearConsole()**

**YDisplayLayer**

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

```
def clearConsole( )
```

**Paramètres :**

**text** le texte à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→consoleOut()displaylayer.consoleOut()****YDisplayLayer**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

```
def consoleOut( text)
```

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

**Paramètres :**

**text** le message à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBar()displaylayer.drawBar()****YDisplayLayer**

Dessine un rectangle plein à une position spécifiée.

```
def drawBar( x1, y1, x2, y2)
```

**Paramètres :**

- x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBitmap()  
displaylayer.drawBitmap()****YDisplayLayer**

Dessine un bitmap à la position spécifiée de la couche.

```
def drawBitmap( x, y, w, bitmap, bgcol)
```

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour laisser les pixels inchangés

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**displaylayer→drawCircle()displaylayer.drawCircle()**

---

**YDisplayLayer**

Dessine un cercle vide à une position spécifiée.

```
def drawCircle( x, y, r)
```

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle
- r** le rayon du cercle, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawDisc()displaylayer.drawDisc()****YDisplayLayer**

Dessine un disque plein à une position spécifiée.

```
def drawDisc( x, y, r)
```

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque
- r** le rayon du disque, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawImage()displaylayer.drawImage()****YDisplayLayer**

Dessine une image GIF à la position spécifiée de la couche.

```
def drawImage( x, y, imagename)
```

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image
- imagename** le nom du fichier GIF à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## displaylayer→drawPixel()displaylayer.drawPixel()

YDisplayLayer

Dessine un pixel unique à une position spécifiée.

```
def drawPixel( x, y)
```

### Paramètres :

- x** la distance en pixels depuis la gauche de la couche
- y** la distance en pixels depuis le haut de la couche

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**displaylayer→drawRect()displaylayer.drawRect()****YDisplayLayer**

Dessine un rectangle vide à une position spécifiée.

```
def drawRect( x1, y1, x2, y2)
```

**Paramètres :**

- x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawText()displaylayer.drawText()****YDisplayLayer**

Affiche un texte à la position spécifiée de la couche.

```
def drawText( x, y, anchor, text)
```

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte
- y** la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte
- anchor** le point d'ancrage du texte, choisi parmi l'énumération Y\_ALIGN: 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** le texte à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**displaylayer**→**get\_display()****YDisplayLayer****displaylayer**→**display()****displaylayer.get\_display()**

---

Retourne l'YDisplay parent.

```
def get_display( )
```

Retourne l'objet YDisplay parent du YDisplayLayer courant.

**Retourne :**

un objet YDisplay

**displaylayer→get\_displayHeight()**

**YDisplayLayer**

**displaylayer→displayHeight()**

**displaylayer.get\_displayHeight()**

---

Retourne la hauteur de l'écran, en pixels.

```
def get_displayHeight( )
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYHEIGHT\_INVALID.

---

**displaylayer→get\_displayWidth()****YDisplayLayer****displaylayer→displayWidth()****displaylayer.get\_displayWidth()**

---

Retourne la largeur de l'écran, en pixels.

```
def get_displayWidth( )
```

**Retourne :**

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYWIDTH\_INVALID.

**displaylayer→get\_layerHeight()**

**YDisplayLayer**

**displaylayer→layerHeight()**

**displaylayer.get\_layerHeight()**

---

Retourne la hauteur des couches affichables, en pixels.

```
def get_layerHeight( )
```

**Retourne :**

un entier représentant la hauteur des couches affichables, en pixels. En cas d'erreur, déclenche une exception ou retourne Y\_LAYERHEIGHT\_INVALID.

---

**displaylayer→get\_layerWidth()**  
**displaylayer→layerWidth()**  
**displaylayer.get\_layerWidth()**

---

**YDisplayLayer**

Retourne la largeur des couches affichables, en pixels.

```
def get_layerWidth( )
```

**Retourne :**

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERWIDTH\_INVALID.

**displaylayer→hide()displaylayer.hide()****YDisplayLayer**

Cache la couche de dessin.

```
def hide( )
```

L'état de la couche est préservé, mais la couche ne sera plus plus affichés à l'écran jusqu'au prochain appel à `unhide( )`. Le fait de cacher la couche améliore les performances de toutes les primitives d'affichage, car il évite de consacrer inutilement des cycles de calcul à afficher les états intermédiaires (technique de double-buffering).

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**displaylayer→lineTo()displaylayer.lineTo()****YDisplayLayer**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

```
def lineTo( x, y )
```

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au point final
- y** la distance en pixels depuis le haut de la couche jusqu'au point final

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→moveTo()displaylayer.moveTo()****YDisplayLayer**

Déplace le point de dessin courant de cette couche à la position spécifiée.

```
def moveTo( x, y)
```

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche de dessin
- y** la distance en pixels depuis le haut de la couche de dessin

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→reset()displaylayer.reset()****YDisplayLayer**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

```
def reset( )
```

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear()`.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectColorPen()**  
**displaylayer.selectColorPen()****YDisplayLayer**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

```
def selectColorPen( color)
```

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

**Paramètres :**

**color** la couleur RGB désirée (sous forme d'entier 24 bits)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**displaylayer→selectEraser()**  
**displaylayer.selectEraser()**

---

**YDisplayLayer**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

```
def selectEraser( )
```

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectFont()displaylayer.selectFont()****YDisplayLayer**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

```
def selectFont( fontname)
```

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

**Paramètres :**

**fontname** le nom du fichier définissant la police de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectGrayPen()**  
**displaylayer.selectGrayPen()**

**YDisplayLayer**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

```
def selectGrayPen( graylevel)
```

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), toute valeur inférieure à 128 conduit à un point noir, et toute valeur supérieure ou égale à 128 devient un point lumineux.

**Paramètres :**

**graylevel** le niveau de gris désiré, de 0 à 255

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setAntialiasingMode()**  
**displaylayer.setAntialiasingMode()****YDisplayLayer**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

```
def setAntialiasingMode( mode)
```

L'anti-aliasing est atténue la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

**Paramètres :**

**mode** true pour activer l'antialiasing, false pour le désactiver.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**displaylayer→setConsoleBackground()**  
**displaylayer.setConsoleBackground()**

---

**YDisplayLayer**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

```
def setConsoleBackground( bgcol)
```

**Paramètres :**

**bgcol** le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleMargins()**  
**displaylayer.setConsoleMargins()**

YDisplayLayer

Configure les marges d'affichage pour la fonction `consoleOut`.

```
def setConsoleMargins( x1, y1, x2, y2)
```

**Paramètres :**

- x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche
- y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure
- x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite
- y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer**→**setConsoleWordWrap()**  
**displaylayer.setConsoleWordWrap()**

**YDisplayLayer**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

```
def setConsoleWordWrap( wordwrap)
```

**Paramètres :**

**wordwrap** `true` pour retourner à la ligne entre les mots seulements, `false` pour retourner à l'extrême droite de chaque ligne.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setLayerPosition()**  
**displaylayer.setLayerPosition()****YDisplayLayer**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

```
def setLayerPosition( x, y, scrollTime)
```

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

**Paramètres :**

- x** la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
- y** la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
- scrollTime** durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**displaylayer→unhide()displaylayer.unhide()**

---

**YDisplayLayer**

Affiche la couche.

```
def unhide( )
```

Affiche a nouveau la couche après la command hide.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.15. Interface de contrôle de l'alimentation

La librairie de programmation Yoctopuce permet de contrôler la source d'alimentation qui doit être utilisée pour les fonctions du module consommant beaucoup de courant. Le module est par ailleurs capable de couper automatiquement l'alimentation externe lorsqu'il détecte que la tension a trop chuté (batterie épuisée).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindDualPower(func)

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

#### yFirstDualPower()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

### Méthodes des objets YDualPower

#### dualpower→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### dualpower→get\_advertisedValue()

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

#### dualpower→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

#### dualpower→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

#### dualpower→get\_extVoltage()

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

#### dualpower→get\_friendlyName()

Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE . NOM_FONCTION`.

#### dualpower→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### dualpower→get\_functionId()

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

#### dualpower→get\_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'alimentation au format `SERIAL . FUNCTIONID`.

#### dualpower→get\_logicalName()

Retourne le nom logique du contrôle d'alimentation.

#### **dualpower→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **dualpower→get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **dualpower→get\_powerControl()**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

#### **dualpower→get\_powerState()**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

#### **dualpower→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **dualpower→isOnline()**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

#### **dualpower→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

#### **dualpower→load(msValidity)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

#### **dualpower→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

#### **dualpower→nextDualPower()**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

#### **dualpower→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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

Modifie le nom logique du contrôle d'alimentation.

#### **dualpower→set\_powerControl(newval)**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

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

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YDualPower.FindDualPower() yFindDualPower()YDualPower.FindDualPower()

YDualPower

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

```
def FindDualPower( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'alimentation soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDualPower.isOnline()` pour tester si le contrôle d'alimentation est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le contrôle d'alimentation sans ambiguïté

### Retourne :

un objet de classe `YDualPower` qui permet ensuite de contrôler le contrôle d'alimentation.



**YDualPower.FirstDualPower()****YDualPower****yFirstDualPower()YDualPower.FirstDualPower()**

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

```
def FirstDualPower( )
```

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

**Retourne :**

un pointeur sur un objet `YDualPower`, correspondant au premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

**dualpower→describe()****dualpower.describe()****YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le contrôle d'alimentation (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

**dualpower**→**get\_advertisedValue()****YDualPower****dualpower**→**advertisedValue()****dualpower.get\_advertisedValue()**

---

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du contrôle d'alimentation (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**dualpower→get\_errorMessage()**

**YDualPower**

**dualpower→errorMessage()**

**dualpower.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

---

**dualpower**→**get\_errorType()****YDualPower****dualpower**→**errorType()****dualpower.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

**dualpower**→**get\_extVoltage()**

**YDualPower**

**dualpower**→**extVoltage()****dualpower.get\_extVoltage()**

---

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

```
def get_extVoltage( )
```

**Retourne :**

un entier représentant la tension mesurée sur l'alimentation de puissance externe, en millivolts

En cas d'erreur, déclenche une exception ou retourne Y\_EXTVOLTAGE\_INVALID.

**dualpower→get\_friendlyName()****YDualPower****dualpower→friendlyName()****dualpower.get\_friendlyName()**

Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du contrôle d'alimentation si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'alimentation (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YDualPower**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID



---

**dualpower**→**get\_functionId()****YDualPower****dualpower**→**functionId()****dualpower.get\_functionId()**

---

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**dualpower**→**get\_hardwareId()**

**YDualPower**

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

---

Retourne l'identifiant matériel unique du contrôle d'alimentation au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'alimentation (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**dualpower→get\_logicalName()****YDualPower****dualpower→logicalName()****dualpower.get\_logicalName()**

---

Retourne le nom logique du contrôle d'alimentation.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du contrôle d'alimentation. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**dualpower**→**get\_module()**

**YDualPower**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

**dualpower→get\_powerControl()****YDualPower****dualpower→powerControl()****dualpower.get\_powerControl()**

---

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
def get_powerControl( )
```

**Retourne :**

une valeur parmi Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT et Y\_POWERCONTROL\_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y\_POWERCONTROL\_INVALID.

**dualpower→get\_powerState()**

**YDualPower**

**dualpower→powerState()**

**dualpower.get\_powerState()**

---

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

def **get\_powerState**( )

**Retourne :**

une valeur parmi Y\_POWERSTATE\_OFF, Y\_POWERSTATE\_FROM\_USB et Y\_POWERSTATE\_FROM\_EXT représentant la source d'alimentation active pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y\_POWERSTATE\_INVALID.

---

**dualpower**→**get\_userdata()****YDualPower****dualpower**→**userData()****dualpower.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **dualpower→isOnline()dualpower.isOnline()**

**YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le contrôle d'alimentation est joignable, `false` sinon



**dualpower→load()dualpower.load()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **dualpower→nextDualPower() dualpower.nextDualPower()**

**YDualPower**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

```
def nextDualPower( )
```

### **Retourne :**

un pointeur sur un objet `YDualPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**dualpower→registerValueCallback()**  
**dualpower.registerValueCallback()**

---

**YDualPower**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**dualpower**→**set\_logicalName()****YDualPower****dualpower**→**setLogicalName()****dualpower.set\_logicalName()**

Modifie le nom logique du contrôle d'alimentation.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**dualpower→set\_powerControl()****YDualPower****dualpower→setPowerControl()****dualpower.set\_powerControl()**

---

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
def set_powerControl( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT et Y\_POWERCONTROL\_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower**→**set\_userdata()**

**YDualPower**

**dualpower**→**setUserData()****dualpower.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.16. Interface de la fonction Files

L'interface de stockage de fichiers permet de stocker des fichiers sur certains modules, par exemple pour personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindFiles(func)

Permet de retrouver un système de fichier d'après un identifiant donné.

#### yFirstFiles()

Commence l'énumération des système de fichier accessibles par la librairie.

### Méthodes des objets YFiles

#### files→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### files→download(pathname)

Télécharge le fichier choisi du filesystem et retourne son contenu.

#### files→download\_async(pathname, callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

#### files→format\_fs()

Rétabli le système de fichier dans on état original, défragmenté.

#### files→get\_advertisedValue()

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

#### files→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### files→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### files→get\_filesCount()

Retourne le nombre de fichiers présents dans le système de fichier.

#### files→get\_freeSpace()

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

#### files→get\_friendlyName()

Retourne un identifiant global du système de fichier au format `NOM_MODULE . NOM_FONCTION`.

#### files→get\_functionDescriptor()

### 3. Reference

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **files**→**get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

#### **files**→**get\_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format `SERIAL . FUNCTIONID`.

#### **files**→**get\_list(pattern)**

Retourne une liste d'objets objet `YFileRecord` qui décrivent les fichiers présents dans le système de fichier.

#### **files**→**get\_logicalName()**

Retourne le nom logique du système de fichier.

#### **files**→**get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **files**→**get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **files**→**get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **files**→**isOnline()**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

#### **files**→**isOnline\_async(callback, context)**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

#### **files**→**load(msValidity)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

#### **files**→**load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

#### **files**→**nextFiles()**

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles()`.

#### **files**→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **files**→**remove(pathname)**

Efface un fichier, spécifié par son path complet, du système de fichier.

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

Modifie le nom logique du système de fichier.

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

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.



**YFiles.FindFiles()****YFiles****yFindFiles()YFiles.FindFiles()**

Permet de retrouver un système de fichier d'après un identifiant donné.

```
def FindFiles( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le système de fichier sans ambiguïté

**Retourne :**

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

## YFiles.FirstFiles()

YFiles

### yFirstFiles()YFiles.FirstFiles()

---

Commence l'énumération des système de fichier accessibles par la librairie.

```
def FirstFiles( )
```

Utiliser la fonction `YFiles.nextFiles( )` pour itérer sur les autres système de fichier.

#### Retourne :

un pointeur sur un objet `YFiles`, correspondant au premier système de fichier accessible en ligne, ou `null` si il n'y a pas de système de fichier disponibles.

**files→describe()files.describe()****YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le système de fichier (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

## files→download(files.download())

YFiles

---

Télécharge le fichier choisi du filesystem et retourne son contenu.

```
def download( pathname)
```

### Paramètres :

**pathname** nom complet du fichier à charger, y compris le chemin d'accès.

### Retourne :

le contenu du fichier chargé sous forme d'objet binaire

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

---

**files**→**format\_fs()****files.format\_fs()****YFiles**

---

Rétabli le système de fichier dans on état original, défragmenté.

```
def format_fs( )
```

entièrement vide. Tous les fichiers précédemment chargés sont irrémédiablement effacés.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files**→**get\_advertisedValue()**

**YFiles**

**files**→**advertisedValue()****files.get\_advertisedValue()**

---

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

---

**files**→**get\_errorMessage()****YFiles****files**→**errorMessage()****files.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

**files**→**get\_errorType()**

**YFiles**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.



**files**→**get\_filesCount()****YFiles****files**→**filesCount()****files.get\_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

```
def get_filesCount( )
```

**Retourne :**

un entier représentant le nombre de fichiers présents dans le système de fichier

En cas d'erreur, déclenche une exception ou retourne `Y_FILESCOUNT_INVALID`.

**files**→**get\_freeSpace()**

**YFiles**

**files**→**freeSpace()****files.get\_freeSpace()**

---

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

```
def get_freeSpace( )
```

**Retourne :**

un entier représentant l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets

En cas d'erreur, déclenche une exception ou retourne `Y_FREESPACE_INVALID`.

---

**files**→**get\_friendlyName()****YFiles****files**→**friendlyName()****files.get\_friendlyName()**

---

Retourne un identifiant global du système de fichier au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du système de fichier si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du système de fichier (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le système de fichier en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YFiles**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**files**→**get\_functionId()****YFiles****files**→**functionId()****files.get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**files**→**get\_hardwareId()**

**YFiles**

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

---

Retourne l'identifiant matériel unique du système de fichier au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**files**→**get\_list()****YFiles****files**→**list()****files.get\_list()**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
def get_list( pattern)
```

**Paramètres :**

**pattern** un filtre optionel sur les noms de fichiers retournés, pouvant contenir des astérisques et des points d'interrogations comme jokers. Si le pattern fourni est vide, tous les fichiers sont retournés.

**Retourne :**

une liste d'objets YFileRecord, contenant le nom complet (y compris le chemin d'accès), la taille en octets et le CRC 32-bit du contenu du fichier.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**files**→**get\_logicalName()**

**YFiles**

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

---

Retourne le nom logique du système de fichier.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du système de fichier. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.



**files**→**get\_module()****YFiles****files**→**module()****files.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**files**→**get\_userdata()**

**YFiles**

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

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**files→isOnline()files.isOnline()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le système de fichier est joignable, `false` sinon

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**files**→**nextFiles()****files.nextFiles()****YFiles**

---

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles()`.

```
def nextFiles( )
```

**Retourne :**

un pointeur sur un objet `YFiles` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**files→registerValueCallback()****YFiles****files.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**files→remove()files.remove()****YFiles**

Efface un fichier, spécifié par son path complet, du système de fichier.

```
def remove( pathname )
```

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occupe. Par contre, la ré-écriture d'un fichier du même nom récupérera dans tout les cas l'espace qui n'aurait éventuellement pas été libéré. Pour s'assurer de libérer la totalité de l'espace du système de fichier, utilisez la fonction `format_fs`.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files**→**set\_logicalName()****YFiles****files**→**setLogicalName()****files.set\_logicalName()**

Modifie le nom logique du système de fichier.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du système de fichier.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**files**→**set\_userdata()****YFiles****files**→**setUserData()****files.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**files→upload()files.upload()****YFiles**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

```
def upload( pathname, content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.17. Interface de la fonction GenericSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindGenericSensor(func)

Permet de retrouver un capteur générique d'après un identifiant donné.

#### yFirstGenericSensor()

Commence l'énumération des capteurs génériques accessibles par la librairie.

### Méthodes des objets YGenericSensor

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### genericsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### genericsensor→get\_advertisedValue()

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

#### genericsensor→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### genericsensor→get\_currentValue()

Retourne la valeur mesurée actuelle.

#### genericsensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### genericsensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### genericsensor→get\_friendlyName()

Retourne un identifiant global du capteur générique au format `NOM_MODULE . NOM_FONCTION`.

#### genericsensor→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### genericsensor→get\_functionId()

Retourne l'identifiant matériel du capteur générique, sans référence au module.

#### genericsensor→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur générique au format SERIAL . FUNCTIONID.

#### **genericsensor→get\_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

#### **genericsensor→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

#### **genericsensor→get\_logicalName()**

Retourne le nom logique du capteur générique.

#### **genericsensor→get\_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

#### **genericsensor→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **genericsensor→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **genericsensor→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

#### **genericsensor→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

#### **genericsensor→get\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **genericsensor→get\_signalRange()**

Retourne la plage de signal électrique utilisée par le capteur.

#### **genericsensor→get\_signalUnit()**

Retourne l'unité du signal électrique utilisée par le capteur.

#### **genericsensor→get\_signalValue()**

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

#### **genericsensor→get\_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

#### **genericsensor→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

#### **genericsensor→get\_valueRange()**

Retourne la plage de valeurs physiques mesurés par le capteur.

#### **genericsensor→isOnline()**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

#### **genericsensor→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

#### **genericsensor→load(msValidity)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

#### **genericsensor→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **genericsensor→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

#### **genericsensor→nextGenericSensor()**

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

**`genericsensor→registerTimedReportCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**`genericsensor→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`genericsensor→set_highestValue(newval)`**

Modifie la mémoire de valeur maximale observée.

**`genericsensor→set_logFrequency(newval)`**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**`genericsensor→set_logicalName(newval)`**

Modifie le nom logique du capteur générique.

**`genericsensor→set_lowestValue(newval)`**

Modifie la mémoire de valeur minimale observée.

**`genericsensor→set_reportFrequency(newval)`**

Modifie la fréquence de notification périodique des valeurs mesurées.

**`genericsensor→set_resolution(newval)`**

Modifie la résolution des valeurs physique mesurées.

**`genericsensor→set_signalRange(newval)`**

Modifie la plage de signal électrique utilisée par le capteur.

**`genericsensor→set_unit(newval)`**

Change l'unité dans laquelle la valeur mesurée est exprimée.

**`genericsensor→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**`genericsensor→set_valueRange(newval)`**

Modifie la plage de valeurs physiques mesurés par le capteur.

**`genericsensor→wait_async(callback, context)`**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YGenericSensor.FindGenericSensor()  
yFindGenericSensor()  
YGenericSensor.FindGenericSensor()****YGenericSensor**

Permet de retrouver un capteur générique d'après un identifiant donné.

```
def FindGenericSensor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur générique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGenericSensor.IsOnline()` pour tester si le capteur générique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur générique sans ambiguïté

**Retourne :**

un objet de classe `YGenericSensor` qui permet ensuite de contrôler le capteur générique.

---

**YGenericSensor.FirstGenericSensor()**  
**yFirstGenericSensor()**  
**YGenericSensor.FirstGenericSensor()**

---

**YGenericSensor**

Commence l'énumération des capteurs génériques accessibles par la librairie.

def **FirstGenericSensor**( )

Utiliser la fonction `YGenericSensor.nextGenericSensor()` pour itérer sur les autres capteurs génériques.

**Retourne :**

un pointeur sur un objet `YGenericSensor`, correspondant au premier capteur générique accessible en ligne, ou `null` si il n'y a pas de capteurs génériques disponibles.

**genericsensor→calibrateFromPoints()  
genericsensor.calibrateFromPoints()****YGenericSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**genericsensor→describe()genericsensor.describe()****YGenericSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le capteur générique (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**genericsensor→get\_advertisedValue()**

**YGenericSensor**

**genericsensor→advertisedValue()**

**genericsensor.get\_advertisedValue()**

---

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

def **get\_advertisedValue**( )

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur générique (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**genericsensor→get\_currentRawValue()****YGenericSensor****genericsensor→currentRawValue()****genericsensor.get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**genericsensor→get\_currentValue()**  
**genericsensor→currentValue()**  
**genericsensor.get\_currentValue()**

---

**YGenericSensor**

Retourne la valeur mesurée actuelle.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**genericsensor→get\_errorMessage()****YGenericSensor****genericsensor→errorMessage()****genericsensor.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

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

**YGenericSensor**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

def **get\_errorType**( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

**genericsensor→get\_friendlyName()****YGenericSensor****genericsensor→friendlyName()****genericsensor.get\_friendlyName()**

Retourne un identifiant global du capteur générique au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur générique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur générique (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur générique en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YGenericSensor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**def get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID



---

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

---

**YGenericSensor**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

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

**YGenericSensor**

---

Retourne l'identifiant matériel unique du capteur générique au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur générique (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**genericsensor→get\_highestValue()****YGenericSensor****genericsensor→highestValue()****genericsensor.get\_highestValue()**

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**genericsensor→get\_logFrequency()**  
**genericsensor→logFrequency()**  
**genericsensor.get\_logFrequency()**

**YGenericSensor**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def **get\_logFrequency**( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**genericsensor→get\_logicalName()****YGenericSensor****genericsensor→logicalName()****genericsensor.get\_logicalName()**

Retourne le nom logique du capteur générique.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur générique. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**genericsensor→get\_lowestValue()**

**YGenericSensor**

**genericsensor→lowestValue()**

**genericsensor.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

def **get\_lowestValue()** ( )

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

---

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

---

**YGenericSensor**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**genericsensor→get\_recordedData()**

**YGenericSensor**

**genericsensor→recordedData()**

**genericsensor.get\_recordedData()**

---

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.



---

**genericsensor→get\_reportFrequency()****YGenericSensor****genericsensor→reportFrequency()****genericsensor.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**genericsensor→get\_resolution()**

**YGenericSensor**

**genericsensor→resolution()**

**genericsensor.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

def **get\_resolution()**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**genericsensor→get\_signalRange()****YGenericSensor****genericsensor→signalRange()****genericsensor.get\_signalRange()**

---

Retourne la plage de signal électrique utilisée par le capteur.

```
def get_signalRange( )
```

**Retourne :**

une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALRANGE\_INVALID.

**genericsensor→get\_signalUnit()**

**YGenericSensor**

**genericsensor→signalUnit()**

**genericsensor.get\_signalUnit()**

---

Retourne l'unité du signal électrique utilisée par le capteur.

def **get\_signalUnit**( )

**Retourne :**

une chaîne de caractères représentant l'unité du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALUNIT\_INVALID.

---

**genericsensor→get\_signalValue()****YGenericSensor****genericsensor→signalValue()****genericsensor.get\_signalValue()**

---

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

```
def get_signalValue( )
```

**Retourne :**

une valeur numérique représentant la valeur mesurée du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALVALUE\_INVALID.

**genericsensor**→**get\_unit()**

**YGenericSensor**

**genericsensor**→**unit()****genericsensor.get\_unit()**

---

Retourne l'unité dans laquelle la mesure est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

---

**genericsensor→get\_userData()**  
**genericsensor→userData()**  
**genericsensor.get\_userData()**

---

**YGenericSensor**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**genericsensor→get\_valueRange()**  
**genericsensor→valueRange()**  
**genericsensor.get\_valueRange()**

---

**YGenericSensor**

Retourne la plage de valeurs physiques mesurés par le capteur.

```
def get_valueRange( )
```

**Retourne :**

une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_VALUERANGE\_INVALID.



---

**genericsensor**→**isOnline()****genericsensor.isOnline()****YGenericSensor**

---

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

**def isOnline( )**

Si les valeurs des attributs en cache du capteur générique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur générique est joignable, `false` sinon

**genericsensor→load()****genericsensor.load()****YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**genericsensor→loadCalibrationPoints()**  
**genericsensor.loadCalibrationPoints()**

---

**YGenericSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor**→**nextGenericSensor()**  
**genericsensor.nextGenericSensor()**

**YGenericSensor**

---

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

def **nextGenericSensor()**

**Retourne :**

un pointeur sur un objet `YGenericSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**genericsensor→registerTimedReportCallback()**  
**genericsensor.registerTimedReportCallback()**

---

**YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**genericsensor→registerValueCallback()**  
**genericsensor.registerValueCallback()****YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

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

**YGenericSensor**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YGenericSensor**

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**genericsensor→set\_logicalName()****YGenericSensor****genericsensor→setLogicalName()****genericsensor.set\_logicalName()**

Modifie le nom logique du capteur générique.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur générique.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YGenericSensor**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_reportFrequency()**  
**genericsensor→setReportFrequency()**  
**genericsensor.set\_reportFrequency()**

**YGenericSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YGenericSensor**

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_signalRange()****YGenericSensor****genericsensor→setSignalRange()****genericsensor.set\_signalRange()**

Modifie la plage de signal électrique utilisée par le capteur.

```
def set_signalRange( newval)
```

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor**→**set\_unit()**

**YGenericSensor**

**genericsensor**→**setUnit()****genericsensor.set\_unit()**

---

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
def set_unit( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_userdata()**  
**genericsensor→setUserData()**  
**genericsensor.set\_userdata()**

**YGenericSensor**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**genericsensor→set\_valueRange()**  
**genericsensor→setValueRange()**  
**genericsensor.set\_valueRange()**

---

**YGenericSensor**

Modifie la plage de valeurs physiques mesurés par le capteur.

```
def set_valueRange( newval)
```

Le changement de plage peut avoir pour effet de bord un changement automatique de la résolution affichée.

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



## 3.18. Interface de la fonction Gyro

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindGyro(func)

Permet de retrouver un gyroscope d'après un identifiant donné.

#### yFirstGyro()

Commence l'énumération des gyroscopes accessibles par la librairie.

### Méthodes des objets YGyro

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### gyro→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### gyro→get\_advertisedValue()

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

#### gyro→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### gyro→get\_currentValue()

Retourne la valeur actuelle de la vitesse angulaire.

#### gyro→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

#### gyro→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

#### gyro→get\_friendlyName()

Retourne un identifiant global du gyroscope au format `NOM_MODULE . NOM_FONCTION`.

#### gyro→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### gyro→get\_functionId()

Retourne l'identifiant matériel du gyroscope, sans référence au module.

#### gyro→get\_hardwareId()

Retourne l'identifiant matériel unique du gyroscope au format `SERIAL . FUNCTIONID`.

**gyro→get\_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

**gyro→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**gyro→get\_logicalName()**

Retourne le nom logique du gyroscope.

**gyro→get\_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

**gyro→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**gyro→get\_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionX()**

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionY()**

Retourne la composante y du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionZ()**

Retourne la composante z du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

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

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**gyro→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**gyro→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**gyro→get\_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

**gyro→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

**gyro→get\_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

#### **gyro→get\_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

#### **gyro→get\_zValue()**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

#### **gyro→isOnline()**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

#### **gyro→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

#### **gyro→load(msValidity)**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

#### **gyro→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

#### **gyro→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

#### **gyro→nextGyro()**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

#### **gyro→registerAnglesCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

#### **gyro→registerQuaternionCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

#### **gyro→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### **gyro→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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

Modifie la mémoire de valeur maximale observée.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du gyroscope.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **gyro→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **gyro→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YGyro.FindGyro() yFindGyro()YGyro.FindGyro()

YGyro

Permet de retrouver un gyroscope d'après un identifiant donné.

```
def FindGyro( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le gyroscope sans ambiguïté

### Retourne :

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

---

**YGyro.FirstGyro()**  
**yFirstGyro()YGyro.FirstGyro()**

---

**YGyro**

Commence l'énumération des gyroscopes accessibles par la librairie.

```
def FirstGyro( )
```

Utiliser la fonction `YGyro.nextGyro( )` pour itérer sur les autres gyroscopes.

**Retourne :**

un pointeur sur un objet `YGyro`, correspondant au premier gyroscope accessible en ligne, ou `null` si il n'y a pas de gyroscopes disponibles.

**gyro→calibrateFromPoints()**  
**gyro.calibrateFromPoints()****YGyro**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→describe()gyro.describe()****YGyro**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le gyroscope (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**gyro**→**get\_advertisedValue()**

**YGyro**

**gyro**→**advertisedValue()****gyro.get\_advertisedValue()**

---

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du gyroscope (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.



---

**gyro→get\_currentRawValue()****YGyro****gyro→currentRawValue()gyro.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**gyro**→**get\_currentValue()**

**YGyro**

**gyro**→**currentValue()****gyro.get\_currentValue()**

---

Retourne la valeur actuelle de la vitesse angulaire.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la vitesse angulaire

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**gyro→get\_errorMessage()****YGyro****gyro→errorMessage()gyro.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

**gyro→get\_errorType()**

**YGyro**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

---

**gyro→get\_friendlyName()****YGyro****gyro→friendlyName()gyro.get\_friendlyName()**

---

Retourne un identifiant global du gyroscope au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du gyroscope si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du gyroscope (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le gyroscope en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YGyro**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**gyro**→**get\_functionId()****YGyro****gyro**→**functionId()****gyro.get\_functionId()**

Retourne l'identifiant matériel du gyroscope, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**gyro**→**get\_hardwareId()**

**YGyro**

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

---

Retourne l'identifiant matériel unique du gyroscope au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.



**gyro→get\_heading()****YGyro****gyro→heading()gyro.get\_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
def get_heading( )
```

L'axe de lacet peut être attribué à n'importe laquelle des directions physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

**Retourne :**

un nombre à virgule correspondant au cap, exprimé en degrés (entre 0 et 360).

**gyro**→**get\_highestValue()**

**YGyro**

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

---

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

---

**gyro→get\_logFrequency()****YGyro****gyro→logFrequency()gyro.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**gyro**→**get\_logicalName()**

**YGyro**

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

---

Retourne le nom logique du gyroscope.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du gyroscope. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**gyro**→**get\_lowestValue()****YGyro****gyro**→**lowestValue()****gyro.get\_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

**gyro**→**get\_module()**

**YGyro**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**gyro**→**get\_pitch()****YGyro****gyro**→**pitch()****gyro.get\_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
def get_pitch( )
```

L'axe de tangage peut être attribué à n'importe laquelle des directions physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

**Retourne :**

un nombre à virgule correspondant à l'assiette, exprimée en degrés (entre -90 et +90).

**gyro**→**get\_quaternionW()**

**YGyro**

**gyro**→**quaternionW()****gyro.get\_quaternionW()**

---

Retourne la composante  $w$  (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

def **get\_quaternionW**( )

**Retourne :**

un nombre à virgule correspondant à la composante  $w$  du quaternion.



---

**gyro**→**get\_quaternionX()****YGyro****gyro**→**quaternionX()****gyro.get\_quaternionX()**

---

Retourne la composante  $x$  du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
def get_quaternionX( )
```

La composante  $x$  est essentiellement corrélée aux rotations sur l'axe de roulis.

**Retourne :**

un nombre à virgule correspondant à la composante  $x$  du quaternion.

**gyro**→**get\_quaternionY()**

**YGyro**

**gyro**→**quaternionY()****gyro.get\_quaternionY()**

---

Retourne la composante  $y$  du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
def get_quaternionY( )
```

La composante  $y$  est essentiellement corrélée aux rotations sur l'axe de tangage.

**Retourne :**

un nombre à virgule correspondant à la composante  $y$  du quaternion.

---

**gyro**→**get\_quaternionZ()****YGyro****gyro**→**quaternionZ()****gyro.get\_quaternionZ()**

---

Retourne la composante *z* du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
def get_quaternionZ( )
```

La composante *z* est essentiellement corrélée aux rotations sur l'axe de lacet.

**Retourne :**

un nombre à virgule correspondant à la composante *z* du quaternion.

**gyro**→**get\_recordedData()****YGyro****gyro**→**recordedData()****gyro.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**gyro→get\_reportFrequency()****YGyro****gyro→reportFrequency()gyro.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**gyro**→**get\_resolution()**

**YGyro**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**gyro**→**get\_roll()****YGyro****gyro**→**roll()****gyro.get\_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
def get_roll( )
```

L'axe de roulis peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

**Retourne :**

un nombre à virgule correspondant à l'inclinaison, exprimée en degrés (entre -180 et +180).

**gyro**→**get\_unit()**

**YGyro**

**gyro**→**unit()****gyro.get\_unit()**

---

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la vitesse angulaire est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.



**gyro→get\_userData()****YGyro****gyro→userData()gyro.get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

**def get\_userData( )**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**gyro**→**get\_xValue()**

**YGyro**

**gyro**→**xValue()****gyro.get\_xValue()**

---

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

```
def get_xValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_XVALUE_INVALID`.

**gyro**→**get\_yValue()**  
**gyro**→**yValue()****gyro.get\_yValue()**

**YGyro**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

```
def get_yValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_YVALUE\_INVALID.

**gyro→get\_zValue()**

**YGyro**

**gyro→zValue()gyro.get\_zValue()**

---

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

```
def get_zValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_ZVALUE\_INVALID.

**gyro→isOnline()gyro.isOnline()****YGyro**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le gyroscope est joignable, `false` sinon

**gyro→load(gyro.load())****YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **gyro→loadCalibrationPoints()** **gyro.loadCalibrationPoints()**

**YGyro**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro**→**nextGyro()****gyro.nextGyro()**

**YGyro**

---

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

def **nextGyro**( )

**Retourne :**

un pointeur sur un objet `YGyro` accessible en ligne, ou `null` lorsque l'énumération est terminée.



---

**gyro→registerAnglesCallback()**  
**gyro.registerAnglesCallback()**

---

**YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
def registerAnglesCallback( callback )
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelés trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter quatre arguments: l'objet YGyro du module qui a tourné, et les valeurs des trois angles roll, pitch et heading en degrés (nombres à virgules).

**gyro→registerQuaternionCallback()**  
**gyro.registerQuaternionCallback()****YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
def registerQuaternionCallback( callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelé trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter cinq arguments: l'objet YGyro du module qui a tourné, et les valeurs des quatre composantes w, x, y et z du quaternion (nombres à virgules).

---

**gyro→registerTimedReportCallback()**  
**gyro.registerTimedReportCallback()**

---

**YGyro**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

**gyro→registerValueCallback()**  
**gyro.registerValueCallback()****YGyro**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**gyro→set\_highestValue()****YGyro****gyro→setHighestValue()gyro.set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_logFrequency()**

**YGyro**

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

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro**→**set\_logicalName()****YGyro****gyro**→**setLogicalName()****gyro.set\_logicalName()**

Modifie le nom logique du gyroscope.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du gyroscope.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro**→**set\_lowestValue()**

**YGyro**

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

---

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**gyro→set\_reportFrequency()**  
**gyro→setReportFrequency()**  
**gyro.set\_reportFrequency()**

**YGyro**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_resolution()**

**YGyro**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro**→**set\_userdata()****YGyro****gyro**→**setUserData()****gyro.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.19. Interface d'un port de Yocto-hub

Les objets YHubPort permettent de contrôler l'alimentation des ports d'un YoctoHub, ainsi que de détecter si un module y est raccordé et lequel. Un YHubPort reçoit toujours automatiquement comme nom logique le numéro de série unique du module Yoctopuce qui y est connecté.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindHubPort(func)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

#### yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

### Méthodes des objets YHubPort

#### hubport→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### hubport→get\_advertisedValue()

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

#### hubport→get\_baudRate()

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

#### hubport→get\_enabled()

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

#### hubport→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

#### hubport→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

#### hubport→get\_friendlyName()

Retourne un identifiant global du port de Yocto-hub au format `NOM_MODULE . NOM_FONCTION`.

#### hubport→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### hubport→get\_functionId()

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

#### hubport→get\_hardwareId()

Retourne l'identifiant matériel unique du port de Yocto-hub au format `SERIAL . FUNCTIONID`.

#### hubport→get\_logicalName()

Retourne le nom logique du port de Yocto-hub.

#### **hubport→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **hubport→get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **hubport→get\_portState()**

Retourne l'état actuel du port de Yocto-hub.

#### **hubport→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **hubport→isOnline()**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

#### **hubport→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

#### **hubport→load(msValidity)**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

#### **hubport→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

#### **hubport→nextHubPort()**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

#### **hubport→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **hubport→set\_enabled(newval)**

Modifie le mode d'activation du port du Yocto-hub.

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

Modifie le nom logique du port de Yocto-hub.

#### **hubport→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YHubPort.FindHubPort()****YHubPort****yFindHubPort()YHubPort.FindHubPort()**

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

```
def FindHubPort( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port de Yocto-hub soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHubPort.isOnline()` pour tester si le port de Yocto-hub est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le port de Yocto-hub sans ambiguïté

**Retourne :**

un objet de classe `YHubPort` qui permet ensuite de contrôler le port de Yocto-hub.

---

**YHubPort.FirstHubPort()**  
**yFirstHubPort()YHubPort.FirstHubPort()**

---

**YHubPort**

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

```
def FirstHubPort( )
```

Utiliser la fonction `YHubPort.nextHubPort( )` pour itérer sur les autres port de Yocto-hub.

**Retourne :**

un pointeur sur un objet `YHubPort`, correspondant au premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

**hubport→describe()hubport.describe()****YHubPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le port de Yocto-hub (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)



---

**hubport→get\_advertisedValue()****YHubPort****hubport→advertisedValue()****hubport.get\_advertisedValue()**

---

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**hubport**→**get\_baudRate()**

**YHubPort**

**hubport**→**baudRate()****hubport.get\_baudRate()**

---

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

```
def get_baudRate( )
```

La valeur par défaut est 1000 kbps, une valeur inférieure révèle des problèmes de communication.

**Retourne :**

un entier représentant la vitesse de transfert utilisée par le port de Yocto-hub, en kbps

En cas d'erreur, déclenche une exception ou retourne Y\_BAUDRATE\_INVALID.

**hubport**→**get\_enabled()****YHubPort****hubport**→**enabled()****hubport.get\_enabled()**

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

```
def get_enabled( )
```

**Retourne :**

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon vrai si le port du Yocto-hub est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

**hubport**→**get\_errorMessage()**

**YHubPort**

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

---

**hubport**→**get\_errorType()****YHubPort****hubport**→**errorType()****hubport.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

**hubport**→**get\_friendlyName()**

**YHubPort**

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

---

Retourne un identifiant global du port de Yocto-hub au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du port de Yocto-hub si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port de Yocto-hub (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

---

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

---

**YHubPort**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

**hubport**→**get\_functionId()**

**YHubPort**

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

---

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.



---

**hubport**→**get\_hardwareId()****YHubPort****hubport**→**hardwareId()****hubport.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du port de Yocto-hub au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port de Yocto-hub (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**hubport**→**get\_logicalName()**

**YHubPort**

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

---

Retourne le nom logique du port de Yocto-hub.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port de Yocto-hub. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**hubport**→**get\_module()****YHubPort****hubport**→**module()****hubport.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**hubport**→**get\_portState()**

**YHubPort**

**hubport**→**portState()****hubport.get\_portState()**

---

Retourne l'état actuel du port de Yocto-hub.

```
def get_portState( )
```

**Retourne :**

une valeur parmi Y\_PORTSTATE\_OFF, Y\_PORTSTATE\_OVRLD, Y\_PORTSTATE\_ON, Y\_PORTSTATE\_RUN et Y\_PORTSTATE\_PROG représentant l'état actuel du port de Yocto-hub

En cas d'erreur, déclenche une exception ou retourne Y\_PORTSTATE\_INVALID.

---

**hubport→get\_userdata()****YHubPort****hubport→userData()hubport.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## hubport→isOnline()hubport.isOnline()

YHubPort

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du port de Yocto-hub sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le port de Yocto-hub est joignable, `false` sinon

**hubport→load()hubport.load()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport**→**nextHubPort()****hubport.nextHubPort()**

**YHubPort**

---

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

```
def nextHubPort( )
```

**Retourne :**

un pointeur sur un objet `YHubPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.



---

**hubport→registerValueCallback()**  
**hubport.registerValueCallback()**

---

**YHubPort**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**hubport**→**set\_enabled()**

**YHubPort**

**hubport**→**setEnabled()****hubport.set\_enabled()**

---

Modifie le mode d'activation du port du Yocto-hub.

```
def set_enabled( newval)
```

Si le port est actif, il sera alimenté. Sinon, l'alimentation du module est coupée.

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon le mode d'activation du port du Yocto-hub

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YHubPort**

Modifie le nom logique du port de Yocto-hub.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du port de Yocto-hub.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport**→**set\_userdata()**

**YHubPort**

**hubport**→**setUserData()****hubport.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.20. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### **yFindHumidity(func)**

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

#### **yFirstHumidity()**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

### Méthodes des objets YHumidity

#### **humidity→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **humidity→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

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

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

#### **humidity→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **humidity→get\_currentValue()**

Retourne la mesure actuelle de l'humidité.

#### **humidity→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### **humidity→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### **humidity→get\_friendlyName()**

Retourne un identifiant global du capteur d'humidité au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **humidity→get\_functionId()**

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

#### **humidity→get\_hardwareId()**

	Retourne l'identifiant matériel unique du capteur d'humidité au format <code>SERIAL.FUNCTIONID</code> .
<b>humidity→get_highestValue()</b>	Retourne la valeur maximale observée pour l'humidité.
<b>humidity→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>humidity→get_logicalName()</b>	Retourne le nom logique du capteur d'humidité.
<b>humidity→get_lowestValue()</b>	Retourne la valeur minimale observée pour l'humidité.
<b>humidity→get_module()</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>humidity→get_module_async(callback, context)</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>humidity→get_recordedData(startTime, endTime)</b>	Retourne un objet <code>DataSet</code> représentant des mesures de ce capteur précédemment enregistrées à l'aide du <code>DataLogger</code> , pour l'intervalle de temps spécifié.
<b>humidity→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>humidity→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>humidity→get_unit()</b>	Retourne l'unité dans laquelle l'humidité est exprimée.
<b>humidity→get_userData()</b>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<b>humidity→isOnline()</b>	Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.
<b>humidity→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.
<b>humidity→load(msValidity)</b>	Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.
<b>humidity→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode <code>calibrateFromPoints</code> .
<b>humidity→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.
<b>humidity→nextHumidity()</b>	Continue l'énumération des capteurs d'humidité commencée à l'aide de <code>yFirstHumidity()</code> .
<b>humidity→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>humidity→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>humidity→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée pour l'humidité.
<b>humidity→set_logFrequency(newval)</b>	

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur d'humidité.

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

Modifie la mémoire de valeur minimale observée pour l'humidité.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**humidity**→**set\_userData**(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YHumidity.FindHumidity() yFindHumidity()YHumidity.FindHumidity()

YHumidity

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

```
def FindHumidity( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur d'humidité soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnline()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

### Retourne :

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.



---

**YHumidity.FirstHumidity()**  
**yFirstHumidity()YHumidity.FirstHumidity()**

---

**YHumidity**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

```
def FirstHumidity( )
```

Utiliser la fonction `YHumidity.nextHumidity( )` pour itérer sur les autres capteurs d'humidité.

**Retourne :**

un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

**humidity→calibrateFromPoints()****YHumidity****humidity.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→describe()humidity.describe()****YHumidity**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le capteur d'humidité (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**humidity→get\_advertisedValue()**

**YHumidity**

**humidity→advertisedValue()**

**humidity.get\_advertisedValue()**

---

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

def **get\_advertisedValue()** ( )

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur d'humidité (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**humidity→get\_currentRawValue()****YHumidity****humidity→currentRawValue()****humidity.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**humidity→get\_currentValue()**

**YHumidity**

**humidity→currentValue()humidity.get\_currentValue()**

---

Retourne la mesure actuelle de l'humidité.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la mesure actuelle de l'humidité

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**humidity→get\_errorMessage()****YHumidity****humidity→errorMessage()****humidity.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

**humidity→get\_errorType()**

**YHumidity**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.



---

**humidity→get\_friendlyName()****YHumidity****humidity→friendlyName()****humidity.get\_friendlyName()**

---

Retourne un identifiant global du capteur d'humidité au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur d'humidité si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur d'humidité (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YHumidity**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**humidity→get\_functionId()****YHumidity****humidity→functionId()humidity.get\_functionId()**

---

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**humidity**→**get\_hardwareId()**

**YHumidity**

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

---

Retourne l'identifiant matériel unique du capteur d'humidité au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur d'humidité (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**humidity→get\_highestValue()****YHumidity****humidity→highestValue()****humidity.get\_highestValue()**

---

Retourne la valeur maximale observée pour l'humidité.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'humidité

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

humidity→get\_logFrequency()

YHumidity

humidity→logFrequency()

humidity.get\_logFrequency()

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def get\_logFrequency( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

**humidity→get\_logicalName()****YHumidity****humidity→logicalName()humidity.get\_logicalName()**

---

Retourne le nom logique du capteur d'humidité.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur d'humidité. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**humidity→get\_lowestValue()**

**YHumidity**

**humidity→lowestValue()humidity.get\_lowestValue()**

---

Retourne la valeur minimale observée pour l'humidité.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'humidité

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.



---

**humidity→get\_module()****YHumidity****humidity→module()humidity.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**humidity→get\_recordedData()****YHumidity****humidity→recordedData()****humidity.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**humidity→get\_reportFrequency()****YHumidity****humidity→reportFrequency()****humidity.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**humidity**→**get\_resolution()**

**YHumidity**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**humidity→get\_unit()****YHumidity****humidity→unit()humidity.get\_unit()**

---

Retourne l'unité dans laquelle l'humidité est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'humidité est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**humidity→get\_userdata()**

**YHumidity**

**humidity→userdata()humidity.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

**humidity→isOnline()humidity.isOnline()****YHumidity**

---

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur d'humidité sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur d'humidité est joignable, `false` sinon

**humidity→load()humidity.load()****YHumidity**

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**humidity→loadCalibrationPoints()**  
**humidity.loadCalibrationPoints()**

---

**YHumidity**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity**→**nextHumidity()****humidity.nextHumidity()**

**YHumidity**

---

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

```
def nextHumidity( )
```

**Retourne :**

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**humidity→registerTimedReportCallback()**  
**humidity.registerTimedReportCallback()**

---

**YHumidity**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

**humidity→registerValueCallback()  
humidity.registerValueCallback()****YHumidity**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

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

**YHumidity**

Modifie la mémoire de valeur maximale observée pour l'humidité.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_logFrequency()**

**YHumidity**

**humidity→setLogFrequency()**

**humidity.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**humidity→set\_logicalName()****YHumidity****humidity→setLogicalName()****humidity.set\_logicalName()**

---

Modifie le nom logique du capteur d'humidité.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur d'humidité.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_lowestValue()**

**YHumidity**

**humidity→setLowestValue()**

**humidity.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour l'humidité.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**humidity→set\_reportFrequency()**  
**humidity→setReportFrequency()**  
**humidity.set\_reportFrequency()**

**YHumidity**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity**→**set\_resolution()**

**YHumidity**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**humidity→set\_userdata()****YHumidity****humidity→setUserData()humidity.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.21. Interface de la fonction Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la led à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindLed(func)

Permet de retrouver une led d'après un identifiant donné.

#### yFirstLed()

Commence l'énumération des leds accessibles par la librairie.

### Méthodes des objets YLed

#### led→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

#### led→get\_advertisedValue()

Retourne la valeur courante de la led (pas plus de 6 caractères).

#### led→get\_blinking()

Retourne le mode de signalisation de la led.

#### led→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

#### led→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

#### led→get\_friendlyName()

Retourne un identifiant global de la led au format `NOM_MODULE . NOM_FONCTION`.

#### led→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### led→get\_functionId()

Retourne l'identifiant matériel de la led, sans référence au module.

#### led→get\_hardwareId()

Retourne l'identifiant matériel unique de la led au format `SERIAL . FUNCTIONID`.

#### led→get\_logicalName()

Retourne le nom logique de la led.

#### led→get\_luminosity()

Retourne l'intensité de la led en pour cent.

#### led→get\_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`led→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`led→get_power()`**

Retourne l'état courant de la led.

**`led→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`led→isOnline()`**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

**`led→isOnline_async(callback, context)`**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

**`led→load(msValidity)`**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

**`led→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

**`led→nextLed()`**

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

**`led→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`led→set_blinking(newval)`**

Modifie le mode de signalisation de la led.

**`led→set_logicalName(newval)`**

Modifie le nom logique de la led.

**`led→set_luminosity(newval)`**

Modifie l'intensité lumineuse de la led (en pour cent).

**`led→set_power(newval)`**

Modifie l'état courant de la led.

**`led→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**`led→wait_async(callback, context)`**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YLed.FindLed()****YLed****yFindLed()YLed.FindLed()**

Permet de retrouver une led d'après un identifiant donné.

```
def FindLed( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la led soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLed.isOnline()` pour tester si la led est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la led sans ambiguïté

**Retourne :**

un objet de classe `YLed` qui permet ensuite de contrôler la led.

**YLed.FirstLed()**  
**yFirstLed()YLed.FirstLed()**

**YLed**

Commence l'énumération des leds accessibles par la librairie.

```
def FirstLed( )
```

Utiliser la fonction `YLed.nextLed( )` pour itérer sur les autres leds.

**Retourne :**

un pointeur sur un objet `YLed`, correspondant à la première led accessible en ligne, ou `null` si il n'y a pas de leds disponibles.

**led→describe()led.describe()****YLed**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant la led (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)



**led**→**get\_advertisedValue()****YLed****led**→**advertisedValue()****led.get\_advertisedValue()**

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**led**→**get\_blinking()**

**YLed**

**led**→**blinking()****led.get\_blinking()**

---

Retourne le mode de signalisation de la led.

```
def get_blinking( )
```

**Retourne :**

une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

En cas d'erreur, déclenche une exception ou retourne Y\_BLINKING\_INVALID.

---

**led**→**get\_errorMessage()****YLed****led**→**errorMessage()****led.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la led.

**led**→**get\_errorType()**

**YLed**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la led.

**led**→**get\_friendlyName()****YLed****led**→**friendlyName()****led.get\_friendlyName()**

Retourne un identifiant global de la led au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la led si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la led (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant la led en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**led**→**get\_functionDescriptor()**

**YLed**

**led**→**functionDescriptor()**

**led.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**led**→**get\_functionId()****YLed****led**→**functionId()****led.get\_functionId()**

Retourne l'identifiant matériel de la led, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la led (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**led**→**get\_hardwareId()**

**YLed**

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

---

Retourne l'identifiant matériel unique de la led au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant la led (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.



**led**→**get\_logicalName()****YLed****led**→**logicalName()****led.get\_logicalName()**

Retourne le nom logique de la led.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de la led. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**led**→**get\_luminosity()**

**YLed**

**led**→**luminosity()****led.get\_luminosity()**

---

Retourne l'intensité de la led en pour cent.

```
def get_luminosity( )
```

**Retourne :**

un entier représentant l'intensité de la led en pour cent

En cas d'erreur, déclenche une exception ou retourne Y\_LUMINOSITY\_INVALID.

**led**→**get\_module()****YLed****led**→**module()****led.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**led**→**get\_power()**

**YLed**

**led**→**power()****led.get\_power()**

---

Retourne l'état courant de la led.

```
def get_power( )
```

**Retourne :**

soit Y\_POWER\_OFF, soit Y\_POWER\_ON, selon l'état courant de la led

En cas d'erreur, déclenche une exception ou retourne Y\_POWER\_INVALID.

**led**→**get\_userData()****YLed****led**→**userData()****led.get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

**def get\_userData( )**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## led→isOnline()led.isOnline()

YLed

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de la led sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si la led est joignable, `false` sinon

**led→load()led.load()****YLed**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led**→**nextLed()****led.nextLed()**

**YLed**

---

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

def **nextLed()**

**Retourne :**

un pointeur sur un objet `YLed` accessible en ligne, ou `null` lorsque l'énumération est terminée.



---

**led→registerValueCallback()****YLed****led.registerValueCallback()**

---

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**led**→**set\_blinking()**

**YLed**

**led**→**setBlinking()****led.set\_blinking()**

---

Modifie le mode de signalisation de la led.

```
def set_blinking( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led**→**set\_logicalName()****YLed****led**→**setLogicalName()****led.set\_logicalName()**

Modifie le nom logique de la led.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la led.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led**→**set\_luminosity()**

**YLed**

**led**→**setLuminosity()****led.set\_luminosity()**

---

Modifie l'intensité lumineuse de la led (en pour cent).

```
def set_luminosity( newval)
```

**Paramètres :**

**newval** un entier représentant l'intensité lumineuse de la led (en pour cent)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led**→**set\_power()****YLed****led**→**setPower()****led.set\_power()**

Modifie l'état courant de la led.

```
def set_power( newval)
```

**Paramètres :**

**newval** soit Y\_POWER\_OFF, soit Y\_POWER\_ON, selon l'état courant de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led**→**set\_userdata()**

**YLed**

**led**→**setUserData()****led.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.22. Interface de la fonction LightSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindLightSensor(func)

Permet de retrouver un capteur de lumière d'après un identifiant donné.

#### yFirstLightSensor()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

### Méthodes des objets YLightSensor

#### lightsensor→calibrate(calibratedVal)

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### lightsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

#### lightsensor→get\_advertisedValue()

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

#### lightsensor→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### lightsensor→get\_currentValue()

Retourne la mesure actuelle de la lumière ambiante.

#### lightsensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

#### lightsensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

#### lightsensor→get\_friendlyName()

Retourne un identifiant global du capteur de lumière au format `NOM_MODULE . NOM_FONCTION`.

#### lightsensor→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### lightsensor→get\_functionId()

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

**lightsensor**→**get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de lumière au format `SERIAL.FUNCTIONID`.

**lightsensor**→**get\_highestValue()**

Retourne la valeur maximale observée pour la lumière ambiante.

**lightsensor**→**get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**lightsensor**→**get\_logicalName()**

Retourne le nom logique du capteur de lumière.

**lightsensor**→**get\_lowestValue()**

Retourne la valeur minimale observée pour la lumière ambiante.

**lightsensor**→**get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**lightsensor**→**get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**lightsensor**→**get\_resolution()**

Retourne la résolution des valeurs mesurées.

**lightsensor**→**get\_unit()**

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

**lightsensor**→**get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**lightsensor**→**isOnline()**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

**lightsensor**→**load(msValidity)**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

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

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

**lightsensor**→**nextLightSensor()**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

**lightsensor**→**registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**lightsensor**→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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



Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur de lumière.

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

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YLightSensor.FindLightSensor() yFindLightSensor()YLightSensor.FindLightSensor()

YLightSensor

Permet de retrouver un capteur de lumière d'après un identifiant donné.

```
def FindLightSensor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de lumière soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLightSensor.isOnline()` pour tester si le capteur de lumière est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de lumière sans ambiguïté

### Retourne :

un objet de classe `YLightSensor` qui permet ensuite de contrôler le capteur de lumière.

**YLightSensor.FirstLightSensor()****YLightSensor****yFirstLightSensor()YLightSensor.FirstLightSensor()**

Commence l'énumération des capteurs de lumière accessibles par la librairie.

```
def FirstLightSensor( )
```

Utiliser la fonction `YLightSensor.nextLightSensor( )` pour itérer sur les autres capteurs de lumière.

**Retourne :**

un pointeur sur un objet `YLightSensor`, correspondant au premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

## lightsensor→calibrate()lightsensor.calibrate()

YLightSensor

Modifie le paramètre de calibration spécifique du capteur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

```
def calibrate( calibratedVal)
```

### Paramètres :

**calibratedVal** la consigne de valeur désirée.

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→calibrateFromPoints()**  
**lightsensor.calibrateFromPoints()****YLightSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→describe()lightsensor.describe()****YLightSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de lumière (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

**lightsensor**→**get\_advertisedValue()****YLightSensor****lightsensor**→**advertisedValue()****lightsensor.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de lumière (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**lightsensor→get\_currentRawValue()**

**YLightSensor**

**lightsensor→currentRawValue()**

**lightsensor.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.



---

**lightsensor→get\_currentValue()****YLightSensor****lightsensor→currentValue()****lightsensor.get\_currentValue()**

---

Retourne la mesure actuelle de la lumière ambiante.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la mesure actuelle de la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**lightsensor→get\_errorMessage()**

**YLightSensor**

**lightsensor→errorMessage()**

**lightsensor.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

**def get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

---

**lightsensor→get\_errorType()****YLightSensor****lightsensor→errorType()lightsensor.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

**lightsensor→get\_friendlyName()**

**YLightSensor**

**lightsensor→friendlyName()**

**lightsensor.get\_friendlyName()**

---

Retourne un identifiant global du capteur de lumière au format `NOM_MODULE . NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de lumière si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de lumière (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

**YLightSensor**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**lightsensor**→**get\_functionId()**

**YLightSensor**

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

---

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

**lightsensor→get\_hardwareId()****YLightSensor****lightsensor→hardwareId()****lightsensor.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du capteur de lumière au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de lumière (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**lightsensor→get\_highestValue()**

**YLightSensor**

**lightsensor→highestValue()**

**lightsensor.get\_highestValue()**

---

Retourne la valeur maximale observée pour la lumière ambiante.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.



---

**lightsensor→get\_logFrequency()****YLightSensor****lightsensor→logFrequency()****lightsensor.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**lightsensor→get\_logicalName()**

**YLightSensor**

**lightsensor→logicalName()**

**lightsensor.get\_logicalName()**

---

Retourne le nom logique du capteur de lumière.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de lumière. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**lightsensor→get\_lowestValue()****YLightSensor****lightsensor→lowestValue()****lightsensor.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la lumière ambiante.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**lightsensor→get\_module()**

**YLightSensor**

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

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**lightsensor→get\_recordedData()****YLightSensor****lightsensor→recordedData()****lightsensor.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

lightsensor→get\_reportFrequency()

YLightSensor

lightsensor→reportFrequency()

lightsensor.get\_reportFrequency()

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

**lightsensor**→**get\_resolution()****YLightSensor****lightsensor**→**resolution()****lightsensor.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

**lightsensor**→**get\_unit()**

**YLightSensor**

**lightsensor**→**unit()****lightsensor.get\_unit()**

---

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la lumière ambiante est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.



---

**lightsensor→get\_userdata()****YLightSensor****lightsensor→userData()lightsensor.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## lightsensor→isOnline()lightsensor.isOnline()

YLightSensor

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de lumière sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de lumière est joignable, `false` sinon

**lightsensor→load()lightsensor.load()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## lightsensor→loadCalibrationPoints()

YLightSensor

### lightsensor.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

#### Paramètres :

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

#### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**lightsensor→nextLightSensor()**  
**lightsensor.nextLightSensor()****YLightSensor**

---

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

```
def nextLightSensor( )
```

**Retourne :**

un pointeur sur un objet `YLightSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**lightsensor→registerTimedReportCallback()  
lightsensor.registerTimedReportCallback()****YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

**lightsensor→registerValueCallback()**  
**lightsensor.registerValueCallback()**

---

**YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**lightsensor→set\_highestValue()**

**YLightSensor**

**lightsensor→setHighestValue()**

**lightsensor.set\_highestValue()**

---

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la lumière ambiante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**lightsensor→set\_logFrequency()****YLightSensor****lightsensor→setLogFrequency()****lightsensor.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logicalName()**

**YLightSensor**

**lightsensor→setLogicalName()**

**lightsensor.set\_logicalName()**

---

Modifie le nom logique du capteur de lumière.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de lumière.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_lowestValue()****YLightSensor****lightsensor→setLowestValue()****lightsensor.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la lumière ambiante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_reportFrequency()**  
**lightsensor→setReportFrequency()**  
**lightsensor.set\_reportFrequency()**

**YLightSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YLightSensor**

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_userdata()**

**YLightSensor**

**lightsensor→setUserData()**

**lightsensor.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.23. Interface de la fonction Magnetometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### **yFindMagnetometer(func)**

Permet de retrouver un magnétomètre d'après un identifiant donné.

#### **yFirstMagnetometer()**

Commence l'énumération des magnétomètres accessibles par la librairie.

### Méthodes des objets YMagnetometer

#### **magnetometer→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **magnetometer→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

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

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

#### **magnetometer→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **magnetometer→get\_currentValue()**

Retourne la valeur actuelle du champ magnétique.

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

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### **magnetometer→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

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

Retourne un identifiant global du magnétomètre au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **magnetometer→get\_functionId()**

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

#### **magnetometer→get\_hardwareId()**

Retourne l'identifiant matériel unique du magnétomètre au format `SERIAL . FUNCTIONID`.

**magnetometer→get\_highestValue()**

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

**magnetometer→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**magnetometer→get\_logicalName()**

Retourne le nom logique du magnétomètre.

**magnetometer→get\_lowestValue()**

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

**magnetometer→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**magnetometer→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**magnetometer→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**magnetometer→get\_unit()**

Retourne l'unité dans laquelle le champ magnétique est exprimée.

**magnetometer→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**magnetometer→get\_xValue()**

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

**magnetometer→get\_yValue()**

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

**magnetometer→get\_zValue()**

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

**magnetometer→isOnline()**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

**magnetometer→load(msValidity)**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

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

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

**magnetometer→nextMagnetometer()**

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

**magnetometer→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.



**magnetometer→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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

Modifie la mémoire de valeur maximale observée.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du magnétomètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**magnetometer→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## **YMagnetometer.FindMagnetometer()** **yFindMagnetometer()** **YMagnetometer.FindMagnetometer()**

**YMagnetometer**

Permet de retrouver un magnétomètre d'après un identifiant donné.

```
def FindMagnetometer( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le magnétomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMagnetometer.isOnline()` pour tester si le magnétomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le magnétomètre sans ambiguïté

**Retourne :**

un objet de classe `YMagnetometer` qui permet ensuite de contrôler le magnétomètre.

**YMagnetometer.FirstMagnetometer()**  
**yFirstMagnetometer()**  
**YMagnetometer.FirstMagnetometer()**

**YMagnetometer**

---

Commence l'énumération des magnétomètres accessibles par la librairie.

def **FirstMagnetometer**( )

Utiliser la fonction `YMagnetometer.nextMagnetometer()` pour itérer sur les autres magnétomètres.

**Retourne :**

un pointeur sur un objet `YMagnetometer`, correspondant au premier magnétomètre accessible en ligne, ou `null` si il n'y a pas de magnétomètres disponibles.

**magnetometer→calibrateFromPoints()**  
**magnetometer.calibrateFromPoints()****YMagnetometer**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→describe()****magnetometer.describe()****YMagnetometer**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le magnétomètre (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

magnetometer→get\_advertisedValue()

YMagnetometer

magnetometer→advertisedValue()

magnetometer.get\_advertisedValue()

---

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du magnétomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**magnetometer→get\_currentRawValue()****YMagnetometer****magnetometer→currentRawValue()****magnetometer.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

magnetometer→get\_currentValue()

YMagnetometer

magnetometer→currentValue()

magnetometer.get\_currentValue()

---

Retourne la valeur actuelle du champ magnétique.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du champ magnétique

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.



---

**magnetometer→get\_errorMessage()****YMagnetometer****magnetometer→errorMessage()****magnetometer.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

**magnetometer→get\_errorType()**

**YMagnetometer**

**magnetometer→errorType()**

**magnetometer.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

def **get\_errorType**( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

---

**magnetometer→get\_friendlyName()****YMagnetometer****magnetometer→friendlyName()****magnetometer.get\_friendlyName()**

---

Retourne un identifiant global du magnétomètre au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du magnétomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du magnétomètre (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le magnétomètre en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**magnetometer→get\_functionDescriptor()**

**YMagnetometer**

**magnetometer→functionDescriptor()**

**magnetometer.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**def get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

**YMagnetometer**

---

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**magnetometer→get\_hardwareId()**

**YMagnetometer**

**magnetometer→hardwareId()**

**magnetometer.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du magnétomètre au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du magnétomètre (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**magnetometer→get\_highestValue()****YMagnetometer****magnetometer→highestValue()****magnetometer.get\_highestValue()**

---

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

magnetometer→get\_logFrequency()

YMagnetometer

magnetometer→logFrequency()

magnetometer.get\_logFrequency()

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def get\_logFrequency( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.



---

**magnetometer→get\_logicalName()****YMagnetometer****magnetometer→logicalName()****magnetometer.get\_logicalName()**

---

Retourne le nom logique du magnétomètre.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du magnétomètre. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

magnetometer→get\_lowestValue()

YMagnetometer

magnetometer→lowestValue()

magnetometer.get\_lowestValue()

---

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

---

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

---

**YMagnetometer**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**magnetometer→get\_recordedData()**

**YMagnetometer**

**magnetometer→recordedData()**

**magnetometer.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**magnetometer→get\_reportFrequency()****YMagnetometer****magnetometer→reportFrequency()****magnetometer.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**magnetometer→get\_resolution()**

**YMagnetometer**

**magnetometer→resolution()**

**magnetometer.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**magnetometer**→**get\_unit()****YMagnetometer****magnetometer**→**unit()****magnetometer.get\_unit()**

---

Retourne l'unité dans laquelle le champ magnétique est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le champ magnétique est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**magnetometer→get\_userdata()**

**YMagnetometer**

**magnetometer→userData()**

**magnetometer.getUserData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.



---

**magnetometer**→**get\_xValue()****YMagnetometer****magnetometer**→**xValue()****magnetometer.get\_xValue()**

---

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

```
def get_xValue( )
```

**Retourne :**

une valeur numérique représentant la composante X du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_XVALUE_INVALID`.

magnetometer→get\_yValue()

YMagnetometer

magnetometer→yValue()magnetometer.get\_yValue()

---

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

```
def get_yValue( )
```

**Retourne :**

une valeur numérique représentant la composante Y du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_YVALUE\_INVALID.

---

**magnetometer**→**get\_zValue()****YMagnetometer****magnetometer**→**zValue()****magnetometer.get\_zValue()**

---

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

```
def get_zValue( )
```

**Retourne :**

une valeur numérique représentant la composante Z du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_ZVALUE_INVALID`.

## magnetometer→isOnline()magnetometer.isOnline()

YMagnetometer

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache du magnétomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le magnétomètre est joignable, `false` sinon

**magnetometer**→**load()****magnetometer.load()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→loadCalibrationPoints()**

**YMagnetometer**

**magnetometer.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**magnetometer**→**nextMagnetometer()**  
**magnetometer.nextMagnetometer()**

---

**YMagnetometer**

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

```
def nextMagnetometer( )
```

**Retourne :**

un pointeur sur un objet `YMagnetometer` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**magnetometer→registerTimedReportCallback()  
magnetometer.registerTimedReportCallback()****YMagnetometer**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.



---

**magnetometer→registerValueCallback()**  
**magnetometer.registerValueCallback()**

---

**YMagnetometer**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

magnetometer→set\_highestValue()

YMagnetometer

magnetometer→setHighestValue()

magnetometer.set\_highestValue()

---

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YMagnetometer**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_logicalName()**

**YMagnetometer**

**magnetometer→setLogicalName()**

**magnetometer.set\_logicalName()**

---

Modifie le nom logique du magnétomètre.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du magnétomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YMagnetometer**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_reportFrequency()**

**YMagnetometer**

**magnetometer→setReportFrequency()**

**magnetometer.set\_reportFrequency()**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**magnetometer→set\_resolution()****YMagnetometer****magnetometer→setResolution()****magnetometer.set\_resolution()**

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_userdata()**

**YMagnetometer**

**magnetometer→setUserData()**

**magnetometer.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser



## 3.24. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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 *

### Méthodes des objets YMeasure

#### **measure→get\_averageValue()**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

#### **measure→get\_maxValue()**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_minValue()**

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_startTimeUTC()**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**measure**→**get\_averageValue()**

**YMeasure**

**measure**→**averageValue()**

**measure.get\_averageValue()**

---

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

```
def get_averageValue( )
```

**Retourne :**

un nombre décimal correspondant à la valeur moyenne observée.

---

**measure→get\_endTimeUTC()****YMeasure****measure→endTimeUTC()measure.get\_endTimeUTC()**

---

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

```
def get_endTimeUTC( )
```

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

**Retourne :**

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

**measure**→**get\_maxValue()**

**YMeasure**

**measure**→**maxValue()****measure.get\_maxValue()**

---

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

```
def get_maxValue( )
```

**Retourne :**

un nombre décimal correspondant à la plus grande valeur observée.

---

**measure**→**get\_minValue()****YMeasure****measure**→**minValue()****measure.get\_minValue()**

---

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

```
def get_minValue( )
```

**Retourne :**

un nombre décimal correspondant à la plus petite valeur observée.

**measure**→**get\_startTimeUTC()**

**YMeasure**

**measure**→**startTimeUTC()**

**measure.get\_startTimeUTC()**

---

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**def get\_startTimeUTC( )**

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

**Retourne :**

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et le début de la mesure.

## 3.25. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindModule(func)

Permet de retrouver un module d'après son numéro de série ou son nom logique.

#### yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

### Méthodes des objets YModule

#### module→describe()

Retourne un court texte décrivant le module.

#### module→download(pathname)

Télécharge le fichier choisi du module et retourne son contenu.

#### module→functionCount()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

#### module→functionId(functionIndex)

Retourne l'identifiant matériel de la *nième* fonction du module.

#### module→functionName(functionIndex)

Retourne le nom logique de la *nième* fonction du module.

#### module→functionValue(functionIndex)

Retourne la valeur publiée par la *nième* fonction du module.

#### module→get\_beacon()

Retourne l'état de la balise de localisation.

#### module→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

#### module→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

#### module→get\_firmwareRelease()

Retourne la version du logiciel embarqué du module.

#### module→get\_hardwareId()

Retourne l'identifiant unique du module.

#### module→get\_icon2d()

### 3. Reference

	Retourne l'icône du module.
<b>module</b> → <b>get_lastLogs()</b>	Retourne une chaîne de caractères contenant les derniers logs du module.
<b>module</b> → <b>get_logicalName()</b>	Retourne le nom logique du module.
<b>module</b> → <b>get_luminosity()</b>	Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).
<b>module</b> → <b>get_persistentSettings()</b>	Retourne l'état courant des réglages persistents du module.
<b>module</b> → <b>get_productId()</b>	Retourne l'identifiant USB du module, préprogrammé en usine.
<b>module</b> → <b>get_productName()</b>	Retourne le nom commercial du module, préprogrammé en usine.
<b>module</b> → <b>get_productRelease()</b>	Retourne le numéro de version matériel du module, préprogrammé en usine.
<b>module</b> → <b>get_rebootCountdown()</b>	Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.
<b>module</b> → <b>get_serialNumber()</b>	Retourne le numéro de série du module, préprogrammé en usine.
<b>module</b> → <b>get_upTime()</b>	Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module.
<b>module</b> → <b>get_usbBandwidth()</b>	Retourne le nombre d'interface USB utilisé par le module.
<b>module</b> → <b>get_usbCurrent()</b>	Retourne le courant consommé par le module sur le bus USB, en milliampères.
<b>module</b> → <b>get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<b>module</b> → <b>isOnline()</b>	Vérifie si le module est joignable, sans déclencher d'erreur.
<b>module</b> → <b>isOnline_async(callback, context)</b>	Vérifie si le module est joignable, sans déclencher d'erreur.
<b>module</b> → <b>load(msValidity)</b>	Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
<b>module</b> → <b>load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
<b>module</b> → <b>nextModule()</b>	Continue l'énumération des modules commencée à l'aide de <code>yFirstModule()</code> .
<b>module</b> → <b>reboot(secBeforeReboot)</b>	Agende un simple redémarrage du module dans un nombre donné de secondes.
<b>module</b> → <b>registerLogCallback(callback)</b>	todo
<b>module</b> → <b>revertFromFlash()</b>	Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.
<b>module</b> → <b>saveToFlash()</b>	



Sauve les réglages courants dans la mémoire non volatile du module.

**module**→**set\_beacon**(**newval**)

Allume ou éteint la balise de localisation du module.

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

Change le nom logique du module.

**module**→**set\_luminosity**(**newval**)

Modifie la luminosité des leds informatives du module.

**module**→**set\_usbBandwidth**(**newval**)

Modifie le nombre d'interface USB utilisé par le module.

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

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**module**→**triggerFirmwareUpdate**(**secBeforeReboot**)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YModule.FindModule() yFindModule()YModule.FindModule()

YModule

Permet de retrouver un module d'après son numéro de série ou son nom logique.

```
def FindModule( func)
```

Cette fonction n'exige pas que le module soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YModule.isOnline()` pour tester si le module est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

### Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

## **YModule.FirstModule()** **yFirstModule()YModule.FirstModule()**

## **YModule**

Commence l'énumération des modules accessibles par la librairie.

```
def FirstModule( )
```

Utiliser la fonction `YModule.nextModule( )` pour itérer sur les autres modules.

### **Retourne :**

un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

**module**→**describe()****module.describe()**

**YModule**

---

Retourne un court texte décrivant le module.

def **describe**( )

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

**Retourne :**

une chaîne de caractères décrivant le module

---

**module**→**download()****module.download()****YModule**

---

Télécharge le fichier choisi du module et retourne son contenu.

```
def download( pathname)
```

**Paramètres :**

**pathname** nom complet du fichier

**Retourne :**

le contenu du fichier chargé

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

**module**→**functionCount()****module.functionCount()**

**YModule**

---

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

def **functionCount**( )

**Retourne :**

le nombre de fonctions sur le module

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**module**→**functionId()****module.functionId()****YModule**

---

Retourne l'identifiant matériel de la *nième* fonction du module.

```
def functionId( functionIndex)
```

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module**→**functionName()****module.functionName()**

**YModule**

---

Retourne le nom logique de la *nième* fonction du module.

def **functionName**( **functionIndex**)

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant au nom logique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.



---

**module**→**functionValue()****module.functionValue()****YModule**

---

Retourne la valeur publiée par la *n*ème fonction du module.

```
def functionValue( functionIndex)
```

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module**→**get\_beacon()**

**YModule**

**module**→**beacon()****module.get\_beacon()**

---

Retourne l'état de la balise de localisation.

```
def get_beacon( )
```

**Retourne :**

soit Y\_BEACON\_OFF, soit Y\_BEACON\_ON, selon l'état de la balise de localisation

En cas d'erreur, déclenche une exception ou retourne Y\_BEACON\_INVALID.

---

**module**→**get\_errorMessage()****YModule****module**→**errorMessage()****module.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

**module**→**get\_errorType()**

**YModule**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

---

**module**→**get\_firmwareRelease()**  
**module**→**firmwareRelease()**  
**module.get\_firmwareRelease()**

---

**YModule**

Retourne la version du logiciel embarqué du module.

```
def get_firmwareRelease( )
```

**Retourne :**

une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne `Y_FIRMWARERELEASE_INVALID`.

**module**→**get\_hardwareId()**

**YModule**

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

---

Retourne l'identifiant unique du module.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

**Retourne :**

une chaîne de caractères identifiant la fonction

---

**module**→**get\_icon2d()****YModule****module**→**icon2d()****module.get\_icon2d()**

---

Retourne l'icône du module.

```
def get_icon2d( )
```

L'icone est au format PNG et a une taille maximale de 1536 octets.

**Retourne :**

un buffer binaire contenant l'icone, au format png.

**module**→**get\_lastLogs()**

**YModule**

**module**→**lastLogs()****module.get\_lastLogs()**

---

Retourne une chaine de caractère contenant les derniers logs du module.

```
def get_lastLogs( )
```

Cette methode retourne les derniers logs qui sont encore stocké dans le module.

**Retourne :**

une chaine de caractère contenant les derniers logs du module.



---

**module**→**get\_logicalName()****YModule****module**→**logicalName()****module.get\_logicalName()**

---

Retourne le nom logique du module.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**module**→**get\_luminosity()**

**YModule**

**module**→**luminosity()****module.get\_luminosity()**

---

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
def get_luminosity( )
```

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne Y\_LUMINOSITY\_INVALID.

---

**module**→**get\_persistentSettings()**  
**module**→**persistentSettings()**  
**module.get\_persistentSettings()**

---

**YModule**

Retourne l'état courant des réglages persistents du module.

```
def get_persistentSettings( )
```

**Retourne :**

une valeur parmi Y\_PERSISTENTSETTINGS\_LOADED, Y\_PERSISTENTSETTINGS\_SAVED et Y\_PERSISTENTSETTINGS\_MODIFIED représentant l'état courant des réglages persistents du module

En cas d'erreur, déclenche une exception ou retourne Y\_PERSISTENTSETTINGS\_INVALID.

**module**→**get\_productId()**

**YModule**

**module**→**productId()****module.get\_productId()**

---

Retourne l'identifiant USB du module, préprogrammé en usine.

```
def get_productId( )
```

**Retourne :**

un entier représentant l'identifiant USB du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_PRODUCTID_INVALID`.

---

**module**→**get\_productName()****YModule****module**→**productName()****module.get\_productName()**

---

Retourne le nom commercial du module, préprogrammé en usine.

```
def get_productName( )
```

**Retourne :**

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTNAME\_INVALID.

**module**→**get\_productRelease()**

**YModule**

**module**→**productRelease()**

**module.get\_productRelease()**

---

Retourne le numéro de version matériel du module, préprogrammé en usine.

```
def get_productRelease( )
```

**Retourne :**

un entier représentant le numéro de version matériel du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTRELEASE\_INVALID.

---

**module→get\_rebootCountdown()****YModule****module→rebootCountdown()****module.get\_rebootCountdown()**

---

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

```
def get_rebootCountdown( )
```

**Retourne :**

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

En cas d'erreur, déclenche une exception ou retourne Y\_REBOOTCOUNTDOWN\_INVALID.

**module**→**get\_serialNumber()**

**YModule**

**module**→**serialNumber()****module.get\_serialNumber()**

---

Retourne le numéro de série du module, préprogrammé en usine.

```
def get_serialNumber( )
```

**Retourne :**

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_SERIALNUMBER\_INVALID.



---

**module**→**get\_upTime()****YModule****module**→**upTime()****module.get\_upTime()**

---

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

```
def get_upTime( )
```

**Retourne :**

un entier représentant le nombre de millisecondes écoulées depuis la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_UPTIME_INVALID`.

**module**→**get\_usbBandwidth()**

**YModule**

**module**→**usbBandwidth()**

**module.get\_usbBandwidth()**

---

Retourne le nombre d'interface USB utilisé par le module.

```
def get_usbBandwidth( )
```

**Retourne :**

soit `Y_USBBANDWIDTH_SIMPLE`, soit `Y_USBBANDWIDTH_DOUBLE`, selon le nombre d'interface USB utilisé par le module

En cas d'erreur, déclenche une exception ou retourne `Y_USBBANDWIDTH_INVALID`.

---

**module**→**get\_usbCurrent()****YModule****module**→**usbCurrent()****module.get\_usbCurrent()**

---

Retourne le courant consommé par le module sur le bus USB, en milliampères.

```
def get_usbCurrent( )
```

**Retourne :**

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

En cas d'erreur, déclenche une exception ou retourne Y\_USBCURRENT\_INVALID.

**module**→**get\_userData()**

**YModule**

**module**→**userData()****module.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

**module→isOnline()module.isOnline()**

---

**YModule**

Vérifie si le module est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le module est joignable, `false` sinon

**module**→**load()****module.load()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**module**→**nextModule()****module.nextModule()****YModule**

---

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

```
def nextModule( )
```

**Retourne :**

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**module→reboot()****module.reboot()****YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
def reboot( secBeforeReboot)
```

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**module**→**registerLogCallback()**  
**module.registerLogCallback()**

---

**YModule**

todo

```
def registerLogCallback( callback)
```

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**module→revertFromFlash()**

**YModule**

**module.revertFromFlash()**

---

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

def **revertFromFlash**( )

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**module→saveToFlash()****module.saveToFlash()****YModule**

---

Sauve les réglages courants dans la mémoire non volatile du module.

```
def saveToFlash( )
```

Attention le nombre total de sauvegardes possibles durant la vie du module est limité (environ 100000 cycles). N'appellez pas cette fonction dans une boucle.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module**→**set\_beacon()**

**YModule**

**module**→**setBeacon()****module.set\_beacon()**

---

Allume ou éteint la balise de localisation du module.

```
def set_beacon( newval)
```

**Paramètres :**

**newval** soit Y\_BEACON\_OFF, soit Y\_BEACON\_ON

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**module**→**set\_logicalName()****YModule****module**→**setLogicalName()****module.set\_logicalName()**

---

Change le nom logique du module.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module**→**set\_luminosity()**

**YModule**

**module**→**setLuminosity()****module.set\_luminosity()**

---

Modifie la luminosité des leds informatives du module.

```
def set_luminosity( newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la luminosité des leds informatives du module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**module**→**set\_usbBandwidth()**  
**module**→**setUsbBandwidth()**  
**module.set\_usbBandwidth()**

---

**YModule**

Modifie le nombre d'interface USB utilisé par le module.

```
def set_usbBandwidth( newval)
```

Vous devez redémarrer le module après avoir changé ce réglage.

**Paramètres :**

**newval** soit Y\_USBBANDWIDTH\_SIMPLE, soit Y\_USBBANDWIDTH\_DOUBLE, selon le nombre d'interface USB utilisé par le module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module**→**set\_userdata()**

**YModule**

**module**→**setUserData()****module.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser



---

**module→triggerFirmwareUpdate()**  
**module.triggerFirmwareUpdate()**

---

**YModule**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

```
def triggerFirmwareUpdate( secBeforeReboot)
```

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.26. Interface de la fonction Network

Les objets YNetwork permettent de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindNetwork(func)

Permet de retrouver une interface réseau d'après un identifiant donné.

#### yFirstNetwork()

Commence l'énumération des interfaces réseau accessibles par la librairie.

### Méthodes des objets YNetwork

#### network→callbackLogin(username, password)

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

#### network→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format TYPE (NAME) = SERIAL . FUNCTIONID.

#### network→get\_adminPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

#### network→get\_advertisedValue()

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

#### network→get\_callbackCredentials()

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

#### network→get\_callbackEncoding()

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

#### network→get\_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

#### network→get\_callbackMethod()

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

#### network→get\_callbackMinDelay()

Retourne l'attente minimale entre deux notifications par callback, en secondes.

#### network→get\_callbackUrl()

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

#### network→get\_discoverable()

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

#### **network→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_friendlyName()**

Retourne un identifiant global de l'interface réseau au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **network→get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

#### **network→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format `SERIAL . FUNCTIONID`.

#### **network→get\_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

#### **network→get\_logicalName()**

Retourne le nom logique de l'interface réseau.

#### **network→get\_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

#### **network→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

#### **network→get\_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

#### **network→get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

#### **network→get\_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

#### **network→get\_secondaryDNS()**

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

#### **network→get\_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

#### **network→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **network→get\_userPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

#### **network→get\_wwwWatchdogDelay()**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

#### **network→isOnline()**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

**network→load(msValidity)**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

**network→nextNetwork()**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

**network→ping(host)**

Ping `str_host` pour vérifier la connexion réseau.

**network→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**network→set\_adminPassword(newval)**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

**network→set\_callbackCredentials(newval)**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

**network→set\_callbackEncoding(newval)**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

**network→set\_callbackMaxDelay(newval)**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

**network→set\_callbackMethod(newval)**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

**network→set\_callbackMinDelay(newval)**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

**network→set\_callbackUrl(newval)**

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

**network→set\_discoverable(newval)**

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

**network→set\_logicalName(newval)**

Modifie le nom logique de l'interface réseau.

**network→set\_primaryDNS(newval)**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

**network→set\_secondaryDNS(newval)**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

**network→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**network→set\_userPassword(newval)**

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

**network→set\_wwwWatchdogDelay(newval)**

Modifie la durée de perte de connexion WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

**network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

**network→useStaticIP(ipAddress, subnetMaskLen, router)**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YNetwork.FindNetwork()****YNetwork****yFindNetwork()YNetwork.FindNetwork()**

Permet de retrouver une interface réseau d'après un identifiant donné.

```
def FindNetwork( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YNetwork.isOnline()` pour tester si l'interface réseau est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'interface réseau sans ambiguïté

**Retourne :**

un objet de classe `YNetwork` qui permet ensuite de contrôler l'interface réseau.

## **YNetwork.FirstNetwork() yFirstNetwork()YNetwork.FirstNetwork()**

**YNetwork**

Commence l'énumération des interfaces réseau accessibles par la librairie.

```
def FirstNetwork( )
```

Utiliser la fonction `YNetwork.nextNetwork( )` pour itérer sur les autres interfaces réseau.

**Retourne :**

un pointeur sur un objet `YNetwork`, correspondant à la première interface réseau accessible en ligne, ou `null` si il n'y a pas de interfaces réseau disponibles.

**network→callbackLogin()network.callbackLogin()****YNetwork**

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

```
def callbackLogin( username, password)
```

Le mot de passe ne sera pas stocké dans le module, mais seulement une version hashée non réversible. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**username** nom d'utilisateur pour s'identifier au callback

**password** mot de passe pour s'identifier au callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**network→describe()network.describe()****YNetwork**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant l'interface réseau (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**network→get\_adminPassword()**

**YNetwork**

**network→adminPassword()**

**network.get\_adminPassword()**

---

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

```
def get_adminPassword( )
```

**Retourne :**

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_ADMINPASSWORD\_INVALID.

---

**network→get\_advertisedValue()****YNetwork****network→advertisedValue()****network.get\_advertisedValue()**

---

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'interface réseau (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**network→get\_callbackCredentials()**

**YNetwork**

**network→callbackCredentials()**

**network.get\_callbackCredentials()**

---

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

```
def get_callbackCredentials( )
```

**Retourne :**

une chaîne de caractères représentant une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKCREDENTIALS\_INVALID.

---

**network→get\_callbackEncoding()****YNetwork****network→callbackEncoding()****network.get\_callbackEncoding()**

---

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

**def get\_callbackEncoding( )**

**Retourne :**

une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKENCODING\_INVALID.

**network→get\_callbackMaxDelay()**

**YNetwork**

**network→callbackMaxDelay()**

**network.get\_callbackMaxDelay()**

---

Retourne l'attente maximale entre deux notifications par callback, en secondes.

```
def get_callbackMaxDelay( )
```

**Retourne :**

un entier représentant l'attente maximale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMAXDELAY\_INVALID.

---

**network**→**get\_callbackMethod()****YNetwork****network**→**callbackMethod()****network.get\_callbackMethod()**

---

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
def get_callbackMethod( )
```

**Retourne :**

une valeur parmi `Y_CALLBACKMETHOD_POST`, `Y_CALLBACKMETHOD_GET` et `Y_CALLBACKMETHOD_PUT` représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMETHOD_INVALID`.

**network→get\_callbackMinDelay()**

**YNetwork**

**network→callbackMinDelay()**

**network.get\_callbackMinDelay()**

---

Retourne l'attente minimale entre deux notifications par callback, en secondes.

```
def get_callbackMinDelay( )
```

**Retourne :**

un entier représentant l'attente minimale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMINDELAY\_INVALID.



**network→get\_callbackUrl()****YNetwork****network→callbackUrl()network.get\_callbackUrl()**

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
def get_callbackUrl( )
```

**Retourne :**

une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKURL\_INVALID.

**network→get\_discoverable()**

**YNetwork**

**network→discoverable()network.get\_discoverable()**

---

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

```
def get_discoverable( )
```

**Retourne :**

soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne Y\_DISCOVERABLE\_INVALID.

---

**network→get\_errorMessage()****YNetwork****network→errorMessage()****network.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

**network**→**get\_errorType()**

**YNetwork**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

---

**network**→**get\_friendlyName()****YNetwork****network**→**friendlyName()****network.get\_friendlyName()**

---

Retourne un identifiant global de l'interface réseau au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'interface réseau si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'interface réseau en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**network**→**get\_functionDescriptor()**

**YNetwork**

**network**→**functionDescriptor()**

**network.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**network**→**get\_functionId()****YNetwork****network**→**functionId()****network.get\_functionId()**

---

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**network**→**get\_hardwareId()**

**YNetwork**

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

---

Retourne l'identifiant matériel unique de l'interface réseau au format `SERIAL.FUNCTIONID`.

**def get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.



**network**→**get\_ipAddress()****YNetwork****network**→**ipAddress()****network.get\_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

```
def get_ipAddress( )
```

Il peut s'agir d'une adresse configurée statiquement, ou d'une adresse reçue par un serveur DHCP.

**Retourne :**

une chaîne de caractères représentant l'adresse IP utilisée par le module Yoctopuce

En cas d'erreur, déclenche une exception ou retourne `Y_IPADDRESS_INVALID`.

**network**→**get\_logicalName()**

**YNetwork**

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

---

Retourne le nom logique de l'interface réseau.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**network→get\_macAddress()****YNetwork****network→macAddress()network.get\_macAddress()**

---

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

```
def get_macAddress( )
```

L'adresse MAC est aussi présente sur un autocollant sur le module, représentée en chiffres et en code-barres.

**Retourne :**

une chaîne de caractères représentant l'adresse MAC de l'interface réseau, unique pour chaque module

En cas d'erreur, déclenche une exception ou retourne Y\_MACADDRESS\_INVALID.

**network**→**get\_module()**

**YNetwork**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

**network**→**get\_poeCurrent()****YNetwork****network**→**poeCurrent()****network.get\_poeCurrent()**

---

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

```
def get_poeCurrent( )
```

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

**Retourne :**

un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne Y\_POECURRENT\_INVALID.

**network→get\_primaryDNS()**

**YNetwork**

**network→primaryDNS()network.get\_primaryDNS()**

---

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
def get_primaryDNS( )
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_PRIMARYDNS\_INVALID.

**network→get\_readiness()****YNetwork****network→readiness()network.get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

```
def get_readiness( )
```

Le niveau zéro (DOWN\_0) signifie qu'aucun support réseau matériel n'a été détecté. Soit il n'y a pas de signal sur le câble réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE\_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme l'existence du SSID configuré. Le niveau 2 (LINK\_2) est atteint lorsque le support matériel du réseau est fonctionnel. Pour une connection réseau filaire, le niveau 2 signifie que le câble est connecté aux deux bouts. Pour une connection à un point d'accès réseau sans fil, il démontre que les paramètres de sécurité configurés sont corrects. Pour une connection sans fil en mode ad-hoc, cela signifie qu'il y a au moins un partenaire sur le réseau ad-hoc. Le niveau 3 (DHCP\_3) est atteint lorsque qu'une adresse IP a été obtenue par DHCP. Le niveau 4 (DNS\_4) est atteint lorsqu'un serveur DNS est joignable par le réseau. Le niveau 5 (WWW\_5) est atteint lorsque la connectivité globale à internet est avérée par l'obtention de l'heure courante sur un serveur NTP.

**Retourne :**

une valeur parmi Y\_READINESS\_DOWN, Y\_READINESS\_EXISTS, Y\_READINESS\_LINKED, Y\_READINESS\_LAN\_OK et Y\_READINESS\_WWW\_OK représentant l'état de fonctionnement atteint par l'interface réseau

En cas d'erreur, déclenche une exception ou retourne Y\_READINESS\_INVALID.

**network**→**get\_router()**

**YNetwork**

**network**→**router()****network.get\_router()**

---

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

```
def get_router( )
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*)

En cas d'erreur, déclenche une exception ou retourne Y\_ROUTER\_INVALID.



---

**network→get\_secondaryDNS()****YNetwork****network→secondaryDNS()****network.get\_secondaryDNS()**

---

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

```
def get_secondaryDNS( )
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_SECONDARYDNS\_INVALID.

**network**→**get\_subnetMask()**

**YNetwork**

**network**→**subnetMask()****network.get\_subnetMask()**

---

Retourne le masque de sous-réseau utilisé par le module.

```
def get_subnetMask( )
```

**Retourne :**

une chaîne de caractères représentant le masque de sous-réseau utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_SUBNETMASK\_INVALID.

---

**network→get\_userdata()****YNetwork****network→userdata()network.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**network→get\_userPassword()**

**YNetwork**

**network→userPassword()**

**network.get\_userPassword()**

---

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

```
def get_userPassword( )
```

**Retourne :**

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_USERPASSWORD\_INVALID.

---

**network**→**get\_wwwWatchdogDelay()****YNetwork****network**→**wwwWatchdogDelay()****network.get\_wwwWatchdogDelay()**

---

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

```
def get_wwwWatchdogDelay( )
```

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW.

**Retourne :**

un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet

En cas d'erreur, déclenche une exception ou retourne `Y_WWWWATCHDOGDELAY_INVALID`.

## **network→isOnline()network.isOnline()**

**YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'interface réseau sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'interface réseau est joignable, `false` sinon

**network→load()network.load()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**nextNetwork()****network.nextNetwork()**

**YNetwork**

---

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

def **nextNetwork**( )

**Retourne :**

un pointeur sur un objet `YNetwork` accessible en ligne, ou `null` lorsque l'énumération est terminée.



**network→ping()network.ping()****YNetwork**

Ping str\_host pour vérifier la connexion réseau.

```
def ping( host)
```

Envoie quatre requêtes ICMP ECHO\_RESPONDER à la cible str\_host depuis le module. Cette méthode retourne une chaîne de caractères avec le résultat des 4 requêtes ICMP ECHO\_RESPONSE.

**Paramètres :**

**host** le nom d'hôte ou l'adresse IP de la cible

**Retourne :**

une chaîne de caractères contenant le résultat du ping.

**network→registerValueCallback()**  
**network.registerValueCallback()****YNetwork**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**network→set\_adminPassword()****YNetwork****network→setAdminPassword()****network.set\_adminPassword()**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

```
def set_adminPassword( newval)
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackCredentials()**  
**network→setCallbackCredentials()**  
**network.set\_callbackCredentials()**

**YNetwork**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

```
def set_callbackCredentials( newval)
```

Le laisser-passer doit être fourni tel que retourné par la fonction `get_callbackCredentials`, sous la forme `username:hash`. La valeur du hash dépend de la méthode d'autorisation implémentée par le callback. Pour une autorisation de type Basic, le hash est le MD5 de la chaîne `username:password`. Pour une autorisation de type Digest, le hash est le MD5 de la chaîne `username:realm:password`. Pour une utilisation simplifiée, utilisez la fonction `callbackLogin`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le laisser-passer pour se connecter à l'adresse de callback

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**network→set\_callbackEncoding()****YNetwork****network→setCallbackEncoding()****network.set\_callbackEncoding()**

---

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
def set_callbackEncoding( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_callbackMaxDelay()**  
**network**→**setCallbackMaxDelay()**  
**network.set\_callbackMaxDelay()**

**YNetwork**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

```
def set_callbackMaxDelay( newval)
```

**Paramètres :**

**newval** un entier représentant l'attente maximale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**network→set\_callbackMethod()****YNetwork****network→setCallbackMethod()****network.set\_callbackMethod()**

---

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
def set_callbackMethod( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_callbackMinDelay()**  
**network**→**setCallbackMinDelay()**  
**network.set\_callbackMinDelay()**

---

**YNetwork**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

```
def set_callbackMinDelay( newval)
```

**Paramètres :**

**newval** un entier représentant l'attente minimale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**network**→**set\_callbackUrl()****YNetwork****network**→**setCallbackUrl()****network.set\_callbackUrl()**

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
def set_callbackUrl( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_discoverable()****YNetwork****network**→**setDiscoverable()****network.set\_discoverable()**

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

```
def set_discoverable( newval)
```

**Paramètres :**

**newval** soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_logicalName()****YNetwork****network**→**setLogicalName()****network.set\_logicalName()**

Modifie le nom logique de l'interface réseau.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_primaryDNS()****YNetwork****network**→**setPrimaryDNS()****network.set\_primaryDNS()**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
def set_primaryDNS( newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_secondaryDNS()****YNetwork****network**→**setSecondaryDNS()****network.set\_secondaryDNS()**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
def set_secondaryDNS( newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_userdata()**

**YNetwork**

**network**→**setUserData()****network.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**network→set\_userPassword()**  
**network→setUserPassword()**  
**network.set\_userPassword()**

**YNetwork**

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

```
def set_userPassword( newval)
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network**→**set\_wwwWatchdogDelay()**

**YNetwork**

**network**→**setWwwWatchdogDelay()**

**network.set\_wwwWatchdogDelay()**

---

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

```
def set_wwwWatchdogDelay( newval)
```

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW. La plus petite durée non-nulle utilisable est 90 secondes.

**Paramètres :**

**newval** un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**network→useDHCP()****network.useDHCP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

```
def useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
```

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

<b>fallbackIpAddr</b>	adresse IP à utiliser si aucun serveur DHCP ne répond
<b>fallbackSubnetMaskLen</b>	longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.
<b>fallbackRouter</b>	adresse de la passerelle à utiliser si aucun serveur DHCP ne répond

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useStaticIP()****network.useStaticIP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

```
def useStaticIP( ipAddress, subnetMaskLen, router)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

<b>ipAddress</b>	adresse IP à utiliser par le module
<b>subnetMaskLen</b>	longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.
<b>router</b>	adresse IP de la passerelle à utiliser ("default gateway")

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.27. contrôle d'OS

L'objet `OsControl` permet de contrôler le système d'exploitation sur lequel tourne un VirtualHub. `OsControl` n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activée au lancement du VirtualHub, avec l'option `-o`.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### `yFindOsControl(func)`

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

#### `yFirstOsControl()`

Commence l'énumération des contrôle d'OS accessibles par la librairie.

### Méthodes des objets `YOsControl`

#### `oscontrol→describe()`

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### `oscontrol→get_advertisedValue()`

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

#### `oscontrol→get_errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

#### `oscontrol→get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

#### `oscontrol→get_friendlyName()`

Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE . NOM_FONCTION`.

#### `oscontrol→get_functionDescriptor()`

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### `oscontrol→get_functionId()`

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

#### `oscontrol→get_hardwareId()`

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL . FUNCTIONID`.

#### `oscontrol→get_logicalName()`

Retourne le nom logique du contrôle d'OS.

#### `oscontrol→get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### `oscontrol→get_module_async(callback, context)`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`oscontrol→get_shutdownCountdown()`**

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

**`oscontrol→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`oscontrol→isOnline()`**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**`oscontrol→isOnline_async(callback, context)`**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**`oscontrol→load(msValidity)`**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**`oscontrol→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**`oscontrol→nextOsControl()`**

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

**`oscontrol→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`oscontrol→set_logicalName(newval)`**

Modifie le nom logique du contrôle d'OS.

**`oscontrol→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**`oscontrol→shutdown(secBeforeShutDown)`**

Agende un arrêt de l'OS dans un nombre donné de secondes.

**`oscontrol→wait_async(callback, context)`**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YOsControl.FindOsControl() yFindOsControl()YOsControl.FindOsControl()

## YOsControl

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

```
def FindOsControl( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'OS soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YOsControl.isOnline()` pour tester si le contrôle d'OS est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

### Retourne :

un objet de classe `YOsControl` qui permet ensuite de contrôler le contrôle d'OS.

**YOsControl.FirstOsControl()**

**YOsControl**

**yFirstOsControl()YOsControl.FirstOsControl()**

---

Commence l'énumération des contrôle d'OS accessibles par la librairie.

```
def FirstOsControl( )
```

Utiliser la fonction `YOsControl.nextOsControl()` pour itérer sur les autres contrôle d'OS.

**Retourne :**

un pointeur sur un objet `YOsControl`, correspondant au premier contrôle d'OS accessible en ligne, ou `null` si il n'y a pas de contrôle d'OS disponibles.

**oscontrol→describe(oscontrol.describe())****YOsControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le contrôle d'OS (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**oscontrol→get\_advertisedValue()**

**YOsControl**

**oscontrol→advertisedValue()**

**oscontrol.get\_advertisedValue()**

---

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

def **get\_advertisedValue**( )

**Retourne :**

une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.



---

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

---

**YOsControl**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

**oscontrol→get\_errorType()**

**YOsControl**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

---

**oscontrol→get\_friendlyName()****YOsControl****oscontrol→friendlyName()****oscontrol.get\_friendlyName()**

---

Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE . NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du contrôle d'OS si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'OS (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YOsControl**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**oscontrol**→**get\_functionId()****YOsControl****oscontrol**→**functionId()****oscontrol.get\_functionId()**

---

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**oscontrol**→**get\_hardwareId()**

**YOsControl**

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

---

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'OS (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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

---

**YOsControl**

Retourne le nom logique du contrôle d'OS.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du contrôle d'OS. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**oscontrol**→**get\_module()**

**YOsControl**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`



---

**oscontrol→get\_shutdownCountdown()****YOsControl****oscontrol→shutdownCountdown()****oscontrol.get\_shutdownCountdown()**

---

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

```
def get_shutdownCountdown( )
```

**Retourne :**

un entier représentant le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé

En cas d'erreur, déclenche une exception ou retourne Y\_SHUTDOWNCOUNTDOWN\_INVALID.

**oscontrol**→**get\_userData()**

**YOsControl**

**oscontrol**→**userData()****oscontrol.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**oscontrol→isOnline()oscontrol.isOnline()****YOsControl**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le contrôle d'OS est joignable, `false` sinon

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**oscontrol→nextOsControl()**  
**oscontrol.nextOsControl()**

---

**YOsControl**

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

```
def nextOsControl( )
```

**Retourne :**

un pointeur sur un objet `YOsControl` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**oscontrol→registerValueCallback()**  
**oscontrol.registerValueCallback()****YOsControl**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**oscontrol→set\_logicalName()****YOsControl****oscontrol→setLogicalName()****oscontrol.set\_logicalName()**

---

Modifie le nom logique du contrôle d'OS.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'OS.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**oscontrol**→**set\_userdata()**

**YOsControl**

**oscontrol**→**setUserData()****oscontrol.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser



---

**oscontrol→shutdown()****oscontrol.shutdown()****YOsControl**

---

Agende un arrêt de l'OS dans un nombre donné de secondes.

```
def shutdown( secBeforeShutDown)
```

**Paramètres :**

**secBeforeShutDown** nombre de secondes avant l'arrêt

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.28. Interface de la fonction Power

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindPower(func)

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

#### yFirstPower()

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

### Méthodes des objets YPower

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### power→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### power→get\_advertisedValue()

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

#### power→get\_cosPhi()

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

#### power→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### power→get\_currentValue()

Retourne la valeur instantanée de la puissance électrique.

#### power→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

#### power→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

#### power→get\_friendlyName()

Retourne un identifiant global du capteur de puissance électrique au format `NOM_MODULE . NOM_FONCTION`.

#### power→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

**`power→get_functionId()`**

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

**`power→get_hardwareId()`**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format `SERIAL.FUNCTIONID`.

**`power→get_highestValue()`**

Retourne la valeur maximale observée pour la puissance électrique.

**`power→get_logFrequency()`**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**`power→get_logicalName()`**

Retourne le nom logique du capteur de puissance électrique.

**`power→get_lowestValue()`**

Retourne la valeur minimale observée pour la puissance électrique.

**`power→get_meter()`**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

**`power→get_meterTimer()`**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

**`power→get_module()`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`power→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`power→get_recordedData(startTime, endTime)`**

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**`power→get_reportFrequency()`**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**`power→get_resolution()`**

Retourne la résolution des valeurs mesurées.

**`power→get_unit()`**

Retourne l'unité dans laquelle la puissance électrique est exprimée.

**`power→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`power→isOnline()`**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

**`power→isOnline_async(callback, context)`**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

**`power→load(msValidity)`**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

**`power→loadCalibrationPoints(rawValues, refValues)`**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

**`power→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

#### **power**→**nextPower()**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

#### **power**→**registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### **power**→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **power**→**reset()**

Réinitialise le compteur d'énergie.

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

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur de puissance électrique.

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

Modifie la mémoire de valeur minimale observée pour la puissance électrique.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs mesurées.

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

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPower.FindPower() yFindPower()YPower.FindPower()

YPower

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

```
def FindPower( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de puissance électrique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance électrique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de puissance électrique sans ambiguïté

### Retourne :

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance électrique.

**YPower.FirstPower()**

**YPower**

**yFirstPower()YPower.FirstPower()**

---

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

```
def FirstPower( )
```

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance électrique.

**Retourne :**

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance électrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance électrique disponibles.

---

**power→calibrateFromPoints()**  
**power.calibrateFromPoints()**

---

**YPower**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues )
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→describe()power.describe()****YPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de puissance électrique (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)



---

**power**→**get\_advertisedValue()****YPower****power**→**advertisedValue()****power.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de puissance électrique (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**power**→**get\_cosPhi()**

**YPower**

**power**→**cosPhi()****power.get\_cosPhi()**

---

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

def **get\_cosPhi()** ( )

**Retourne :**

une valeur numérique représentant le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA)

En cas d'erreur, déclenche une exception ou retourne Y\_COSPHI\_INVALID.

---

**power**→**get\_currentRawValue()****YPower****power**→**currentRawValue()****power.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**power**→**get\_currentValue()**

**YPower**

**power**→**currentValue()****power.get\_currentValue()**

---

Retourne la valeur instantanée de la puissance électrique.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur instantanée de la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**power**→**get\_errorMessage()****YPower****power**→**errorMessage()****power.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

**power**→**get\_errorType()**

**YPower**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

---

**power**→**get\_friendlyName()****YPower****power**→**friendlyName()****power.get\_friendlyName()**

---

Retourne un identifiant global du capteur de puissance électrique au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de puissance électrique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de puissance électrique (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**power**→**get\_functionDescriptor()**

**YPower**

**power**→**functionDescriptor()**

**power.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID



---

**power**→**get\_functionId()****YPower****power**→**functionId()****power.get\_functionId()**

---

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**power**→**get\_hardwareId()**

**YPower**

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

---

Retourne l'identifiant matériel unique du capteur de puissance électrique au format `SERIAL.FUNCTIONID`.

def **get\_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de puissance électrique (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**power**→**get\_highestValue()****YPower****power**→**highestValue()****power.get\_highestValue()**

---

Retourne la valeur maximale observée pour la puissance électrique.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**power**→**get\_logFrequency()**

**YPower**

**power**→**logFrequency()****power.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def **get\_logFrequency()** ( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

**power**→**get\_logicalName()****YPower****power**→**logicalName()****power.get\_logicalName()**

---

Retourne le nom logique du capteur de puissance électrique.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de puissance électrique. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**power**→**get\_lowestValue()**

**YPower**

**power**→**lowestValue()****power.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la puissance électrique.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

---

**power**→**get\_meter()****YPower****power**→**meter()****power.get\_meter()**

---

Retourne la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée.

```
def get_meter( )
```

Ce compteur est réinitialisé à chaque démarrage du module.

**Retourne :**

une valeur numérique représentant la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée

En cas d'erreur, déclenche une exception ou retourne Y\_METER\_INVALID.

**power**→**get\_meterTimer()**

**YPower**

**power**→**meterTimer()****power.get\_meterTimer()**

---

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

```
def get_meterTimer( )
```

**Retourne :**

un entier représentant le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_METERTIMER\_INVALID.



**power**→**get\_module()****YPower****power**→**module()****power.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**power**→**get\_recordedData()****YPower****power**→**recordedData()****power.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**power→get\_reportFrequency()****YPower****power→reportFrequency()****power.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**power**→**get\_resolution()**

**YPower**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**power**→**get\_unit()****YPower****power**→**unit()****power.get\_unit()**

---

Retourne l'unité dans laquelle la puissance électrique est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la puissance électrique est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**power**→**get\_userdata()**

**YPower**

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

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**power→isOnline()power.isOnline()****YPower**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de puissance électrique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de puissance électrique est joignable, `false` sinon

**power→load()****power.load()****YPower**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**power→loadCalibrationPoints()**  
**power.loadCalibrationPoints()**

---

**YPower**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power**→**nextPower()****power.nextPower()**

**YPower**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

```
def nextPower( )
```

**Retourne :**

un pointeur sur un objet `YPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**power→registerTimedReportCallback()**  
**power.registerTimedReportCallback()**

---

**YPower**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

**power→registerValueCallback()****YPower****power.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**power→reset()****power.reset()****YPower**

Réinitialise le compteur d'énergie.

```
def reset( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power**→**set\_highestValue()**

**YPower**

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

---

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la puissance électrique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**power**→**set\_logFrequency()****YPower****power**→**setLogFrequency()****power.set\_logFrequency()**

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power**→**set\_logicalName()**

**YPower**

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

---

Modifie le nom logique du capteur de puissance electrique.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de puissance electrique.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**power**→**set\_lowestValue()****YPower****power**→**setLowestValue()****power.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée pour la puissance électrique.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la puissance électrique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power**→**set\_reportFrequency()**  
**power**→**setReportFrequency()**  
**power.set\_reportFrequency()**

**YPower**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**power**→**set\_resolution()****YPower****power**→**setResolution()****power.set\_resolution()**

---

Modifie la résolution des valeurs mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power**→**set\_userdata()**

**YPower**

**power**→**setUserData()****power.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.29. Interface de la fonction Pressure

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### **yFindPressure(func)**

Permet de retrouver un capteur de pression d'après un identifiant donné.

#### **yFirstPressure()**

Commence l'énumération des capteurs de pression accessibles par la librairie.

### Méthodes des objets YPressure

#### **pressure→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **pressure→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

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

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

#### **pressure→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **pressure→get\_currentValue()**

Retourne la mesure actuelle de la pression.

#### **pressure→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### **pressure→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### **pressure→get\_friendlyName()**

Retourne un identifiant global du capteur de pression au format `NOM_MODULE . NOM_FONCTION`.

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

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### **pressure→get\_functionId()**

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

#### **pressure→get\_hardwareId()**

	Retourne l'identifiant matériel unique du capteur de pression au format <code>SERIAL.FUNCTIONID</code> .
<b>pressure</b> → <b>get_highestValue()</b>	Retourne la valeur maximale observée pour la pression.
<b>pressure</b> → <b>get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>pressure</b> → <b>get_logicalName()</b>	Retourne le nom logique du capteur de pression.
<b>pressure</b> → <b>get_lowestValue()</b>	Retourne la valeur minimale observée pour la pression.
<b>pressure</b> → <b>get_module()</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>pressure</b> → <b>get_module_async(callback, context)</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>pressure</b> → <b>get_recordedData(startTime, endTime)</b>	Retourne un objet <code>DataSet</code> représentant des mesures de ce capteur précédemment enregistrées à l'aide du <code>DataLogger</code> , pour l'intervalle de temps spécifié.
<b>pressure</b> → <b>get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>pressure</b> → <b>get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>pressure</b> → <b>get_unit()</b>	Retourne l'unité dans laquelle la pression est exprimée.
<b>pressure</b> → <b>get_userData()</b>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<b>pressure</b> → <b>isOnline()</b>	Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.
<b>pressure</b> → <b>isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.
<b>pressure</b> → <b>load(msValidity)</b>	Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.
<b>pressure</b> → <b>loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode <code>calibrateFromPoints</code> .
<b>pressure</b> → <b>load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.
<b>pressure</b> → <b>nextPressure()</b>	Continue l'énumération des capteurs de pression commencée à l'aide de <code>yFirstPressure()</code> .
<b>pressure</b> → <b>registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>pressure</b> → <b>registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>pressure</b> → <b>set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée pour la pression.
<b>pressure</b> → <b>set_logFrequency(newval)</b>	

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur de pression.

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

Modifie la mémoire de valeur minimale observée pour la pression.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**pressure**→**set\_userData**(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPressure.FindPressure() yFindPressure()YPressure.FindPressure()

YPressure

Permet de retrouver un capteur de pression d'après un identifiant donné.

```
def FindPressure( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnline()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de pression sans ambiguïté

### Retourne :

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.



---

**YPressure.FirstPressure()**  
**yFirstPressure()YPressure.FirstPressure()**

---

**YPressure**

Commence l'énumération des capteurs de pression accessibles par la librairie.

```
def FirstPressure( )
```

Utiliser la fonction `YPressure.nextPressure( )` pour itérer sur les autres capteurs de pression.

**Retourne :**

un pointeur sur un objet `YPressure`, correspondant au premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

**pressure→calibrateFromPoints()  
pressure.calibrateFromPoints()****YPressure**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→describe()pressure.describe()****YPressure**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de pression (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**pressure**→**get\_advertisedValue()**

**YPressure**

**pressure**→**advertisedValue()**

**pressure.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

def **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de pression (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

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

---

**YPressure**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**pressure**→**get\_currentValue()**

**YPressure**

**pressure**→**currentValue()****pressure.get\_currentValue()**

---

Retourne la mesure actuelle de la pression.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la mesure actuelle de la pression

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

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

---

**YPressure**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

**pressure**→**get\_errorType()**

**YPressure**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.



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

**YPressure**

---

Retourne un identifiant global du capteur de pression au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de pression si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de pression (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de pression en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YPressure**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**pressure**→**get\_functionId()****YPressure****pressure**→**functionId()****pressure.get\_functionId()**

---

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de pression (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pressure**→**get\_hardwareId()**

**YPressure**

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

---

Retourne l'identifiant matériel unique du capteur de pression au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de pression (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de pression (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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

---

**YPressure**

Retourne la valeur maximale observée pour la pression.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la pression

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

**pressure→get\_logFrequency()**

**YPressure**

**pressure→logFrequency()**

**pressure.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**def get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

**pressure→get\_logicalName()****YPressure****pressure→logicalName()pressure.get\_logicalName()**

---

Retourne le nom logique du capteur de pression.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de pression. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pressure**→**get\_lowestValue()**

**YPressure**

**pressure**→**lowestValue()****pressure.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la pression.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la pression

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.



---

**pressure**→**get\_module()****YPressure****pressure**→**module()****pressure.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**pressure**→**get\_recordedData()****YPressure****pressure**→**recordedData()****pressure.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**pressure→get\_reportFrequency()****YPressure****pressure→reportFrequency()****pressure.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**pressure**→**get\_resolution()**

**YPressure**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**pressure**→**get\_unit()****YPressure****pressure**→**unit()****pressure.get\_unit()**

---

Retourne l'unité dans laquelle la pression est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la pression est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**pressure**→**get\_userdata()**

**YPressure**

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

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

**pressure**→**isOnline()****pressure.isOnline()****YPressure**

---

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de pression sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de pression est joignable, `false` sinon

**pressure→load()pressure.load()****YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**pressure→loadCalibrationPoints()**  
**pressure.loadCalibrationPoints()**

---

**YPressure**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure**→**nextPressure()****pressure.nextPressure()**

**YPressure**

---

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

def **nextPressure()**

**Retourne :**

un pointeur sur un objet `YPressure` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**pressure→registerTimedReportCallback()  
pressure.registerTimedReportCallback()**

---

**YPressure**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

**pressure→registerValueCallback()  
pressure.registerValueCallback()****YPressure**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

**pressure**→**set\_highestValue()****YPressure****pressure**→**setHighestValue()****pressure.set\_highestValue()**

---

Modifie la mémoire de valeur maximale observée pour la pression.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la pression

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

---

**YPressure**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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

---

**YPressure**

Modifie le nom logique du capteur de pression.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de pression.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure**→**set\_lowestValue()**

**YPressure**

**pressure**→**setLowestValue()**

**pressure.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour la pression.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la pression

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**pressure→set\_reportFrequency()**  
**pressure→setReportFrequency()**  
**pressure.set\_reportFrequency()**

**YPressure**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure**→**set\_resolution()**

**YPressure**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**pressure**→**set\_userdata()****YPressure****pressure**→**setUserData()****pressure.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.30. Interface de la fonction Pwm

La librairie de programmation Yoctopuce permet simplement de configurer, démarrer et arrêter le PWM.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindPwmOutput(func)

Permet de retrouver un PWM d'après un identifiant donné.

#### yFirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

### Méthodes des objets YPwmOutput

#### pwmoutput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

#### pwmoutput→dutyCycleMove(target, ms\_duration)

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

#### pwmoutput→get\_advertisedValue()

Retourne la valeur courante du PWM (pas plus de 6 caractères).

#### pwmoutput→get\_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

#### pwmoutput→get\_dutyCycleAtPowerOn()

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

#### pwmoutput→get\_enabled()

Retourne l'état de fonctionnement du PWM.

#### pwmoutput→get\_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

#### pwmoutput→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

#### pwmoutput→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

#### pwmoutput→get\_frequency()

Retourne la fréquence du PWM en Hz.

#### pwmoutput→get\_friendlyName()

Retourne un identifiant global du PWM au format `NOM_MODULE . NOM_FONCTION`.

#### pwmoutput→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### `pwmoutput→get_functionId()`

Retourne l'identifiant matériel du PWM, sans référence au module.

#### `pwmoutput→get_hardwareId()`

Retourne l'identifiant matériel unique du PWM au format `SERIAL.FUNCTIONID`.

#### `pwmoutput→get_logicalName()`

Retourne le nom logique du PWM.

#### `pwmoutput→get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### `pwmoutput→get_module_async(callback, context)`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### `pwmoutput→get_period()`

Retourne la période du PWM en millisecondes.

#### `pwmoutput→get_pulseDuration()`

Retourne la longueur d'une impulsion du PWM en millisecondes.

#### `pwmoutput→get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### `pwmoutput→isOnline()`

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

#### `pwmoutput→isOnline_async(callback, context)`

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

#### `pwmoutput→load(msValidity)`

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

#### `pwmoutput→load_async(msValidity, callback, context)`

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

#### `pwmoutput→nextPwmOutput()`

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

#### `pwmoutput→pulseDurationMove(ms_target, ms_duration)`

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

#### `pwmoutput→registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### `pwmoutput→set_dutyCycle(newval)`

Modifie le duty cycle du PWM, en pour cents.

#### `pwmoutput→set_dutyCycleAtPowerOn(newval)`

Modifie le duty cycle du PWM au démarrage du module.

#### `pwmoutput→set_enabled(newval)`

Démarre ou arrête le PWM.

#### `pwmoutput→set_enabledAtPowerOn(newval)`

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

#### `pwmoutput→set_frequency(newval)`

Modifie la fréquence du PWM.

#### `pwmoutput→set_logicalName(newval)`

Modifie le nom logique du PWM.

#### `pwmoutput→set_period(newval)`

Modifie la période du PWM.

### 3. Reference

---

#### **pwmoutput→set\_pulseDuration(newval)**

Modifie la longueur des impulsion du PWM, en millisecondes.

#### **pwmoutput→set\_userdata(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPwmOutput.FindPwmOutput() yFindPwmOutput()YPwmOutput.FindPwmOutput()

YPwmOutput

Permet de retrouver un PWM d'après un identifiant donné.

```
def FindPwmOutput( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le PWM soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmOutput.isOnline()` pour tester si le PWM est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le PWM sans ambiguïté

### Retourne :

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

**YPwmOutput.FirstPwmOutput()**

**YPwmOutput**

**yFirstPwmOutput()****YPwmOutput.FirstPwmOutput()**

---

Commence l'énumération des PWM accessibles par la librairie.

```
def FirstPwmOutput( )
```

Utiliser la fonction `YPwmOutput.nextPwmOutput( )` pour itérer sur les autres PWM.

**Retourne :**

un pointeur sur un objet `YPwmOutput`, correspondant au premier PWM accessible en ligne, ou `null` si il n'y a pas de PWM disponibles.



**pwmoutput→describe()****pwmoutput.describe()****YPwmOutput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le PWM (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**pwmoutput→dutyCycleMove()**  
**pwmoutput.dutyCycleMove()****YPwmOutput**

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

```
def dutyCycleMove( target, ms_duration)
```

**Paramètres :**

**target** nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)  
**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**pwmoutput→get\_advertisedValue()****YPwmOutput****pwmoutput→advertisedValue()****pwmoutput.get\_advertisedValue()**

---

Retourne la valeur courante du PWM (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du PWM (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**pwmoutput→get\_dutyCycle()**

**YPwmOutput**

**pwmoutput→dutyCycle()****pwmoutput.get\_dutyCycle()**

---

Retourne le duty cycle du PWM, en pour cents.

```
def get_dutyCycle( )
```

**Retourne :**

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne Y\_DUTYCYCLE\_INVALID.

---

**pwmoutput→get\_dutyCycleAtPowerOn()**  
**pwmoutput→dutyCycleAtPowerOn()**  
**pwmoutput.get\_dutyCycleAtPowerOn()**

---

**YPwmOutput**

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

```
def get_dutyCycleAtPowerOn( )
```

**Retourne :**

une valeur numérique représentant le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

En cas d'erreur, déclenche une exception ou retourne `Y_DUTYCYCLEATPOWERON_INVALID`.

**pwmoutput→get\_enabled()**

**YPwmOutput**

**pwmoutput→enabled()****pwmoutput.get\_enabled()**

---

Retourne l'état de fonctionnement du PWM.

```
def get_enabled( )
```

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du PWM

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

---

**pwmoutput→get\_enabledAtPowerOn()**  
**pwmoutput→enabledAtPowerOn()**  
**pwmoutput.get\_enabledAtPowerOn()**

---

**YPwmOutput**

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

```
def get_enabledAtPowerOn( )
```

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état de fonctionnement du PWM à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

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

---

**YPwmOutput**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du PWM.



---

**pwmoutput→get\_errorType()****YPwmOutput****pwmoutput→errorType()pwmoutput.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

**pwmoutput→get\_frequency()**

**YPwmOutput**

**pwmoutput→frequency()pwmoutput.get\_frequency()**

---

Retourne la fréquence du PWM en Hz.

```
def get_frequency( )
```

**Retourne :**

un entier représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne Y\_FREQUENCY\_INVALID.

---

**pwmoutput→get\_friendlyName()****YPwmOutput****pwmoutput→friendlyName()****pwmoutput.get\_friendlyName()**

---

Retourne un identifiant global du PWM au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du PWM si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du PWM (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le PWM en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YPwmOutput**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**pwmoutput**→**get\_functionId()****YPwmOutput****pwmoutput**→**functionId()****pwmoutput.get\_functionId()**

---

Retourne l'identifiant matériel du PWM, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pwmoutput→get\_hardwareId()**

**YPwmOutput**

**pwmoutput→hardwareId()**

**pwmoutput.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du PWM au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du PWM (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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

---

**YPwmOutput**

Retourne le nom logique du PWM.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du PWM. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pwmoutput→get\_module()**

**YPwmOutput**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`



---

**pwmoutput→get\_period()****YPwmOutput****pwmoutput→period()pwmoutput.get\_period()**

---

Retourne la période du PWM en millisecondes.

```
def get_period( )
```

**Retourne :**

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PERIOD\_INVALID.

**pwmoutput→get\_pulseDuration()**  
**pwmoutput→pulseDuration()**  
**pwmoutput.get\_pulseDuration()**

---

**YPwmOutput**

Retourne la longueur d'une impulsion du PWM en millisecondes.

```
def get_pulseDuration( )
```

**Retourne :**

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PULSEDURATION\_INVALID.

---

**pwmoutput→get\_userdata()****YPwmOutput****pwmoutput→userdata()pwmoutput.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **pwmoutput→isOnline()pwmoutput.isOnline()**

**YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du PWM sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le PWM est joignable, `false` sinon

**pwmoutput→load()pwmoutput.load()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→nextPwmOutput()**  
**pwmoutput.nextPwmOutput()**

---

**YPwmOutput**

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

```
def nextPwmOutput( )
```

**Retourne :**

un pointeur sur un objet `YPwmOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**pwmoutput→pulseDurationMove()**  
**pwmoutput.pulseDurationMove()**

---

**YPwmOutput**

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

```
def pulseDurationMove( ms_target, ms_duration)
```

N'importe quel changement de fréquence, duty cycle, période ou encore de longueur d'impulsion annulera tout processus de transition en cours.

**Paramètres :**

- ms\_target** nouvelle longueur des impulsions à la fin de la transition (nombre flottant, représentant la longueur en millisecondes)
- ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→registerValueCallback()**  
**pwmoutput.registerValueCallback()****YPwmOutput**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.



**pwmoutput→set\_dutyCycle()**  
**pwmoutput→setDutyCycle()**  
**pwmoutput.set\_dutyCycle()**

**YPwmOutput**

Modifie le duty cycle du PWM, en pour cents.

```
def set_dutyCycle( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM, en pour cents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_dutyCycleAtPowerOn()**

**YPwmOutput**

**pwmoutput→setDutyCycleAtPowerOn()**

**pwmoutput.set\_dutyCycleAtPowerOn()**

---

Modifie le duty cycle du PWM au démarrage du module.

```
def set_dutyCycleAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM au démarrage du module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**pwmoutput→set\_enabled()****YPwmOutput****pwmoutput→setEnabled()****pwmoutput.set\_enabled()**

---

Démarre ou arrête le PWM.

```
def set_enabled( newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_enabledAtPowerOn()**  
**pwmoutput→setEnabledAtPowerOn()**  
**pwmoutput.set\_enabledAtPowerOn()**

**YPwmOutput**

---

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

```
def set_enabledAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du fonctionnement du PWM à la mise sous tension du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_frequency()**  
**pwmoutput→setFrequency()**  
**pwmoutput.set\_frequency()**

**YPwmOutput**

Modifie la fréquence du PWM.

```
def set_frequency( newval)
```

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

**Paramètres :**

**newval** un entier représentant la fréquence du PWM

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_logicalName()**

**YPwmOutput**

**pwmoutput→setLogicalName()**

**pwmoutput.set\_logicalName()**

---

Modifie le nom logique du PWM.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du PWM.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**pwmoutput→set\_period()****YPwmOutput****pwmoutput→setPeriod()pwmoutput.set\_period()**

---

Modifie la période du PWM.

```
def set_period( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la période du PWM

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_pulseDuration()**  
**pwmoutput→setPulseDuration()**  
**pwmoutput.set\_pulseDuration()**

---

**YPwmOutput**

Modifie la longueur des impulsion du PWM, en millisecondes.

```
def set_pulseDuration( newval)
```

Attention la longueur d'un impulsion ne peut pas être plus grande que la période, dans la cas contraire, la longueur sera automatiquement tronqué à la période.

**Paramètres :**

**newval** une valeur numérique représentant la longueur des impulsion du PWM, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**pwmoutput→set\_userdata()****YPwmOutput****pwmoutput→setUserData()****pwmoutput.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.31. Interface de la fonction PwmPowerSource

La librairie de programmation Yoctopuce permet de configurer la source de tension utilisée par tous les PWM situés sur un même module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindPwmPowerSource(func)

Permet de retrouver une source de tension d'après un identifiant donné.

#### yFirstPwmPowerSource()

Commence l'énumération des Source de tension accessibles par la librairie.

### Méthodes des objets YPwmPowerSource

#### pwmpowersource→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### pwmpowersource→get\_advertisedValue()

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

#### pwmpowersource→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

#### pwmpowersource→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

#### pwmpowersource→get\_friendlyName()

Retourne un identifiant global de la source de tension au format `NOM_MODULE . NOM_FONCTION`.

#### pwmpowersource→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### pwmpowersource→get\_functionId()

Retourne l'identifiant matériel de la source de tension, sans référence au module.

#### pwmpowersource→get\_hardwareId()

Retourne l'identifiant matériel unique de la source de tension au format `SERIAL . FUNCTIONID`.

#### pwmpowersource→get\_logicalName()

Retourne le nom logique de la source de tension.

#### pwmpowersource→get\_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### pwmpowersource→get\_module\_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`pwmpowersource→get_powerMode()`**

Retourne la source de tension utilisé par tous les PWM du même module.

**`pwmpowersource→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`pwmpowersource→isOnline()`**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**`pwmpowersource→isOnline_async(callback, context)`**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**`pwmpowersource→load(msValidity)`**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

**`pwmpowersource→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

**`pwmpowersource→nextPwmPowerSource()`**

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

**`pwmpowersource→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`pwmpowersource→set_logicalName(newval)`**

Modifie le nom logique de la source de tension.

**`pwmpowersource→set_powerMode(newval)`**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

**`pwmpowersource→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**`pwmpowersource→wait_async(callback, context)`**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPwmPowerSource.FindPwmPowerSource() yFindPwmPowerSource() YPwmPowerSource.FindPwmPowerSource()

YPwmPowerSource

Permet de retrouver une source de tension d'après un identifiant donné.

```
def FindPwmPowerSource( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmPowerSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

### Retourne :

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

**YPwmPowerSource.FirstPwmPowerSource()**  
**yFirstPwmPowerSource()**  
**YPwmPowerSource.FirstPwmPowerSource()**

**YPwmPowerSource**

---

Commence l'énumération des Source de tension accessibles par la librairie.

```
def FirstPwmPowerSource( )
```

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource( )` pour itérer sur les autres Source de tension.

**Retourne :**

un pointeur sur un objet `YPwmPowerSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de Source de tension disponibles.

**pwmpowersource→describe()**  
**pwmpowersource.describe()****YPwmPowerSource**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

```
une chaîne de caractères décrivant la source de tension (ex:  
Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)
```

---

**pwmpowersource→get\_advertisedValue()**

**YPwmPowerSource**

**pwmpowersource→advertisedValue()**

**pwmpowersource.get\_advertisedValue()**

---

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**pwmpowersource→get\_errorMessage()**

**YPwmPowerSource**

**pwmpowersource→errorMessage()**

**pwmpowersource.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.



---

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

---

**YPwmPowerSource**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

**pwmpowersource→get\_friendlyName()**

**YPwmPowerSource**

**pwmpowersource→friendlyName()**

**pwmpowersource.get\_friendlyName()**

---

Retourne un identifiant global de la source de tension au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la source de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la source de tension (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant la source de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

---

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

---

**YPwmPowerSource**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

---

**YPwmPowerSource**

Retourne l'identifiant matériel de la source de tension, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la source de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

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

---

**YPwmPowerSource**

Retourne l'identifiant matériel unique de la source de tension au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la source de tension (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant la source de tension (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**pwmpowersource→get\_logicalName()**

**YPwmPowerSource**

**pwmpowersource→logicalName()**

**pwmpowersource.get\_logicalName()**

---

Retourne le nom logique de la source de tension.

def **get\_logicalName()** ( )

**Retourne :**

une chaîne de caractères représentant le nom logique de la source de tension. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

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

---

**YPwmPowerSource**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**pwmpowersource→get\_powerMode()**

**YPwmPowerSource**

**pwmpowersource→powerMode()**

**pwmpowersource.get\_powerMode()**

---

Retourne la source de tension utilisé par tous les PWM du même module.

def **get\_powerMode**( )

**Retourne :**

une valeur parmi Y\_POWERMODE\_USB\_5V, Y\_POWERMODE\_USB\_3V, Y\_POWERMODE\_EXT\_V et Y\_POWERMODE\_OPNDRN représentant la source de tension utilisé par tous les PWM du même module

En cas d'erreur, déclenche une exception ou retourne Y\_POWERMODE\_INVALID.



---

**pwmpowersource→get\_userdata()****YPwmPowerSource****pwmpowersource→userData()****pwmpowersource.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pwmpowersource**→**isOnline()**  
**pwmpowersource.isOnline()**

**YPwmPowerSource**

---

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**def isOnline( )**

Si les valeurs des attributs en cache de la source de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si la source de tension est joignable, `false` sinon

---

**pwmpowersource→load()pwmpowersource.load()****YPwmPowerSource**

---

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource**→**nextPwmPowerSource()**  
**pwmpowersource.nextPwmPowerSource()**

**YPwmPowerSource**

---

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

def **nextPwmPowerSource()**

**Retourne :**

un pointeur sur un objet `YPwmPowerSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**pwmpowersource→registerValueCallback()**  
**pwmpowersource.registerValueCallback()**

---

**YPwmPowerSource**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pwmpowersource→set\_logicalName()**

**YPwmPowerSource**

**pwmpowersource→setLogicalName()**

**pwmpowersource.set\_logicalName()**

---

Modifie le nom logique de la source de tension.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la source de tension.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→set\_powerMode()**  
**pwmpowersource→setPowerMode()**  
**pwmpowersource.set\_powerMode()**

**YPwmPowerSource**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

```
def set_powerMode( newval)
```

Le PWM peut aussi en mode open drain, dans ce code il tire activement la ligne à zéro volts. Attention ce paramètre est commun à tous les PWM du module, si vous changez le valeur de ce paramètre, tous les PWM situés sur le même module seront affectés. Si vous souhaitez que le changement de ce paramètre soit conservé après un redémarrage du module, n'oubliez pas d'appeler la méthode `saveToFlash()`.

**Paramètres :**

**newval** une valeur parmi `Y_POWERMODE_USB_5V`, `Y_POWERMODE_USB_3V`, `Y_POWERMODE_EXT_V` et `Y_POWERMODE_OPNDRN` représentant le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→set\_userdata()**

**YPwmPowerSource**

**pwmpowersource→setUserData()**

**pwmpowersource.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser



## 3.32. Interface du quaternion

La class YQt de la librairie Yoctopuce permet d'accéder à l'estimation de l'orientation tridimensionnelle du Yocto-3D sous forme d'un quaternion. Il n'est en général pas nécessaire d'y accéder directement, la classe YGyro offrant une abstraction de plus haut niveau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindQt(func)

Permet de retrouver un élément de quaternion d'après un identifiant donné.

#### yFirstQt()

Commence l'énumération des éléments de quaternion accessibles par la librairie.

### Méthodes des objets YQt

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### qt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### qt→get\_advertisedValue()

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

#### qt→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### qt→get\_currentValue()

Retourne la valeur actuelle de la coordonnée.

#### qt→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

#### qt→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

#### qt→get\_friendlyName()

Retourne un identifiant global de l'élément de quaternion au format `NOM_MODULE . NOM_FONCTION`.

#### qt→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### qt→get\_functionId()

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

**qt→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'élément de quaternion au format `SERIAL.FUNCTIONID`.

**qt→get\_highestValue()**

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

**qt→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**qt→get\_logicalName()**

Retourne le nom logique de l'élément de quaternion.

**qt→get\_lowestValue()**

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

**qt→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**qt→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**qt→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**qt→get\_unit()**

Retourne l'unité dans laquelle la coordonnée est exprimée.

**qt→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**qt→isOnline()**

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

**qt→load(msValidity)**

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

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

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

**qt→nextQt()**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

**qt→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**qt→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**qt→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

**qt→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**qt→set\_logicalName(newval)**

Modifie le nom logique de l'élément de quaternion.

**qt→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**qt→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**qt→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**qt→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userData.

**qt→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YQt.FindQt()****YQt****yFindQt()YQt.FindQt()**

Permet de retrouver un élément de quaternion d'après un identifiant donné.

```
def FindQt( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnline()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

**Retourne :**

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

**YQt.FirstQt()****YQt****yFirstQt()YQt.FirstQt()**

Commence l'énumération des éléments de quaternion accessibles par la librairie.

```
def FirstQt( )
```

Utiliser la fonction `YQt.nextQt( )` pour itérer sur les autres éléments de quaternion.

**Retourne :**

un pointeur sur un objet `YQt`, correspondant au premier élément de quaternion accessible en ligne, ou `null` si il n'y a pas de éléments de quaternion disponibles.

**qt→calibrateFromPoints()qt.calibrateFromPoints()****YQt**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→describe()qt.describe()****YQt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'élément de quaternion (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**qt→get\_advertisedValue()**

**YQt**

**qt→advertisedValue()qt.get\_advertisedValue()**

---

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.



---

**qt**→**get\_currentRawValue()****YQt****qt**→**currentRawValue()****qt.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**qt→get\_currentValue()**

**YQt**

**qt→currentValue()qt.get\_currentValue()**

---

Retourne la valeur actuelle de la coordonnée.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la coordonnée

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**qt→get\_errorMessage()****YQt****qt→errorMessage()qt.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

**qt→get\_errorType()**

**YQt**

**qt→errorType()qt.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

**qt→get\_friendlyName()****YQt****qt→friendlyName()qt.get\_friendlyName()**

Retourne un identifiant global de l'élément de quaternion au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'élément de quaternion si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'élément de quaternion (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**qt→get\_functionDescriptor()**

**YQt**

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

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**qt**→**get\_functionId()****YQt****qt**→**functionId()****qt.get\_functionId()**

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**qt**→**get\_hardwareId()**

**YQt**

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

---

Retourne l'identifiant matériel unique de l'élément de quaternion au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'élément de quaternion (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.



**qt→get\_highestValue()****YQt****qt→highestValue()qt.get\_highestValue()**

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**qt→get\_logFrequency()**

**YQt**

**qt→logFrequency()qt.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def **get\_logFrequency()** ( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**qt**→**get\_logicalName()****YQt****qt**→**logicalName()****qt.get\_logicalName()**

Retourne le nom logique de l'élément de quaternion.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'élément de quaternion. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**qt→get\_lowestValue()**

**YQt**

**qt→lowestValue()qt.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

def **get\_lowestValue**( )

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**qt→get\_module()****YQt****qt→module()qt.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**qt→get\_recordedData()****YQt****qt→recordedData()qt.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**qt→get\_reportFrequency()****YQt****qt→reportFrequency()qt.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**qt**→**get\_resolution()**

**YQt**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.



**qt→get\_unit()****YQt****qt→unit()qt.get\_unit()**

Retourne l'unité dans laquelle la coordonnée est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la coordonnée est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**qt→get\_userdata()**

**YQt**

**qt→userdata()qt.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**qt→isOnline()qt.isOnline()****YQt**

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'élément de quaternion est joignable, `false` sinon

**qt→load()qt.load()****YQt**

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→loadCalibrationPoints()qt.loadCalibrationPoints()****YQt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt**→**nextQt()****qt.nextQt()**

**YQt**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

```
def nextQt( )
```

**Retourne :**

un pointeur sur un objet `YQt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**qt→registerTimedReportCallback()**  
**qt.registerTimedReportCallback()**

---

**YQt**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**qt→registerValueCallback()**  
**qt.registerValueCallback()**

YQt

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.



**qt→set\_highestValue()****YQt****qt→setHighestValue()qt.set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logFrequency()****YQt****qt→setLogFrequency()qt.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logicalName()****YQt****qt→setLogicalName()qt.set\_logicalName()**

Modifie le nom logique de l'élément de quaternion.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'élément de quaternion.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_lowestValue()**

**YQt**

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

---

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**qt→set\_reportFrequency()****YQt****qt→setReportFrequency()qt.set\_reportFrequency()**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt**→**set\_resolution()**

**YQt**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_userdata()****YQt****qt→setUserData()qt.set\_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

### 3.33. Interface de la fonction Horloge Temps Real

La fonction RealTimeClock fourni la date et l'heure courante de manière persistante, même en cas de coupure de courant de plusieurs jours. Elle est le fondement des fonctions de réveil automatique implémentées par le WakeUpScheduler. L'heure courante peut représenter aussi bien une heure locale qu'une heure UTC, mais aucune adaptation automatique n'est fait au changement d'heure été/hiver.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
<b>yFindRealTimeClock(func)</b> Permet de retrouver une horloge d'après un identifiant donné.
<b>yFirstRealTimeClock()</b> Commence l'énumération des horloge accessibles par la librairie.
Méthodes des objets YRealTimeClock
<b>realtimeclock→describe()</b> Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format TYPE ( NAME ) = SERIAL . FUNCTIONID.
<b>realtimeclock→get_advertisedValue()</b> Retourne la valeur courante de l'horloge (pas plus de 6 caractères).
<b>realtimeclock→get_dateTime()</b> Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"
<b>realtimeclock→get_errorMessage()</b> Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.
<b>realtimeclock→get_errorType()</b> Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.
<b>realtimeclock→get_friendlyName()</b> Retourne un identifiant global de l'horloge au format NOM_MODULE . NOM_FONCTION.
<b>realtimeclock→get_functionDescriptor()</b> Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>realtimeclock→get_functionId()</b> Retourne l'identifiant matériel de l'horloge, sans référence au module.
<b>realtimeclock→get_hardwareId()</b> Retourne l'identifiant matériel unique de l'horloge au format SERIAL . FUNCTIONID.
<b>realtimeclock→get_logicalName()</b> Retourne le nom logique de l'horloge.
<b>realtimeclock→get_module()</b>



Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`realtimeclock→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`realtimeclock→get_timeSet()`**

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

**`realtimeclock→get_unixTime()`**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

**`realtimeclock→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`realtimeclock→get_utcOffset()`**

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**`realtimeclock→isOnline()`**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**`realtimeclock→isOnline_async(callback, context)`**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**`realtimeclock→load(msValidity)`**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**`realtimeclock→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**`realtimeclock→nextRealTimeClock()`**

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

**`realtimeclock→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`realtimeclock→set_logicalName(newval)`**

Modifie le nom logique de l'horloge.

**`realtimeclock→set_unixTime(newval)`**

Modifie l'heure courante.

**`realtimeclock→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**`realtimeclock→set_utcOffset(newval)`**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**`realtimeclock→wait_async(callback, context)`**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## **YRealTimeClock.FindRealTimeClock() yFindRealTimeClock() YRealTimeClock.FindRealTimeClock()**

**YRealTimeClock**

Permet de retrouver une horloge d'après un identifiant donné.

```
def FindRealTimeClock( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnline()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'horloge sans ambiguïté

**Retourne :**

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

**YRealTimeClock.FirstRealTimeClock()**  
**yFirstRealTimeClock()**  
**YRealTimeClock.FirstRealTimeClock()**

**YRealTimeClock**

---

Commence l'énumération des horloge accessibles par la librairie.

```
def FirstRealTimeClock( )
```

Utiliser la fonction `YRealTimeClock.nextRealTimeClock( )` pour itérer sur les autres horloge.

**Retourne :**

un pointeur sur un objet `YRealTimeClock`, correspondant à la première horloge accessible en ligne, ou `null` si il n'y a pas de horloge disponibles.

**realtimeclock→describe()realtimeclock.describe()****YRealTimeClock**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format  
`TYPE(NAME)=SERIAL.FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'horloge (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

**realtimeclock→get\_advertisedValue()****YRealTimeClock****realtimeclock→advertisedValue()****realtimeclock.get\_advertisedValue()**

---

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**realtimeclock→get\_dateTime()**

**YRealTimeClock**

**realtimeclock→dateTime()**

**realtimeclock.get\_dateTime()**

---

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

```
def get_dateTime( )
```

**Retourne :**

une chaîne de caractères représentant l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

En cas d'erreur, déclenche une exception ou retourne Y\_DATETIME\_INVALID.

---

**realtimeclock→get\_errorMessage()****YRealTimeClock****realtimeclock→errorMessage()****realtimeclock.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

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

**YRealTimeClock**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

**def get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.



---

**realtimeclock→get\_friendlyName()****YRealTimeClock****realtimeclock→friendlyName()****realtimeclock.get\_friendlyName()**

---

Retourne un identifiant global de l'horloge au format `NOM_MODULE . NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'horloge si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'horloge (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'horloge en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YRealTimeClock**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

def **get\_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

**YRealTimeClock**

---

Retourne l'identifiant matériel de l'horloge, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

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

---

**YRealTimeClock**

Retourne l'identifiant matériel unique de l'horloge au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'horloge (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**realtimeclock→get\_logicalName()****YRealTimeClock****realtimeclock→logicalName()****realtimeclock.get\_logicalName()**

---

Retourne le nom logique de l'horloge.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'horloge. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**realtimeclock→get\_module()**

**YRealTimeClock**

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

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

---

**realtimeclock**→**get\_timeSet()****YRealTimeClock****realtimeclock**→**timeSet()****realtimeclock.get\_timeSet()**

---

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

```
def get_timeSet( )
```

**Retourne :**

soit Y\_TIMESET\_FALSE, soit Y\_TIMESET\_TRUE, selon vrai si l'horloge à été mise à l'heure, sinon faux

En cas d'erreur, déclenche une exception ou retourne Y\_TIMESET\_INVALID.

**realtimeclock→get\_unixTime()**  
**realtimeclock→unixTime()**  
**realtimeclock.get\_unixTime()**

**YRealTimeClock**

---

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

```
def get_unixTime( )
```

**Retourne :**

un entier représentant l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970)

En cas d'erreur, déclenche une exception ou retourne Y\_UNIXTIME\_INVALID.



---

**realtimeclock→get\_userdata()****YRealTimeClock****realtimeclock→userData()****realtimeclock.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**realtimeclock→get\_utcOffset()**

**YRealTimeClock**

**realtimeclock→utcOffset()**

**realtimeclock.get\_utcOffset()**

---

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
def get_utcOffset( )
```

**Retourne :**

un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne Y\_UTCOffset\_INVALID.

---

**realtimeclock→isOnline()realtimeclock.isOnline()****YRealTimeClock**

---

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'horloge est joignable, `false` sinon

**realtimeclock→load()realtimeclock.load()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**realtimeclock→nextRealTimeClock()**  
**realtimeclock.nextRealTimeClock()**

---

**YRealTimeClock**

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

```
def nextRealTimeClock( )
```

**Retourne :**

un pointeur sur un objet `YRealTimeClock` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**realtimeclock→registerValueCallback()  
realtimeclock.registerValueCallback()****YRealTimeClock**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

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

**YRealTimeClock**

Modifie le nom logique de l'horloge.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'horloge.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→set\_unixTime()**  
**realtimeclock→setUnixTime()**  
**realtimeclock.set\_unixTime()**

---

**YRealTimeClock**

Modifie l'heure courante.

```
def set_unixTime( newval)
```

L'heure est passée au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970). Si l'heure UTC est connue, l'attribut utcOffset sera automatiquement ajusté en fonction de l'heure configurée.

**Paramètres :**

**newval** un entier représentant l'heure courante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**realtimeclock→set\_userdata()****YRealTimeClock****realtimeclock→setUserData()****realtimeclock.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**realtimeclock→set\_utcOffset()**

**YRealTimeClock**

**realtimeclock→setUtcOffset()**

**realtimeclock.set\_utcOffset()**

---

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
def set_utcOffset( newval)
```

Le décalage est automatiquement arrondi au quart d'heure le plus proche. Si l'heure UTC est connue, l'heure courante sera automatiquement adaptée en fonction du décalage choisi.

**Paramètres :**

**newval** un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.34. Configuration du référentiel

Cette classe permet de configurer l'orientation dans laquelle le Yocto-3D est utilisé, afin que les fonctions d'orientation relatives au plan de la surface terrestre utilisent le référentiel approprié. La classe offre aussi un processus de recalibration tridimensionnel des capteurs, permettant de compenser les variations locales de l'accélération terrestre et d'améliorer la précision des capteurs d'inclinaisons.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### **yFindRefFrame(func)**

Permet de retrouver un référentiel d'après un identifiant donné.

#### **yFirstRefFrame()**

Commence l'énumération des référentiels accessibles par la librairie.

### Méthodes des objets YRefFrame

#### **refframe→cancel3DCalibration()**

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

#### **refframe→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

#### **refframe→get\_3DCalibrationHint()**

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

#### **refframe→get\_3DCalibrationLogMsg()**

Retourne le dernier message de log produit par le processus de calibration.

#### **refframe→get\_3DCalibrationProgress()**

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.

#### **refframe→get\_3DCalibrationStage()**

Retourne l'index de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

#### **refframe→get\_3DCalibrationStageProgress()**

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

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

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

#### **refframe→get\_bearing()**

### 3. Reference

	Retourne le cap de référence utilisé par le compas.
<b>refframe</b> → <b>get_errorMessage()</b>	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.
<b>refframe</b> → <b>get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.
<b>refframe</b> → <b>get_friendlyName()</b>	Retourne un identifiant global du référentiel au format NOM_MODULE . NOM_FONCTION.
<b>refframe</b> → <b>get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>refframe</b> → <b>get_functionId()</b>	Retourne l'identifiant matériel du référentiel, sans référence au module.
<b>refframe</b> → <b>get_hardwareId()</b>	Retourne l'identifiant matériel unique du référentiel au format SERIAL . FUNCTIONID.
<b>refframe</b> → <b>get_logicalName()</b>	Retourne le nom logique du référentiel.
<b>refframe</b> → <b>get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>refframe</b> → <b>get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>refframe</b> → <b>get_mountOrientation()</b>	Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.
<b>refframe</b> → <b>get_mountPosition()</b>	Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.
<b>refframe</b> → <b>get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>refframe</b> → <b>isOnline()</b>	Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.
<b>refframe</b> → <b>isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.
<b>refframe</b> → <b>load(msValidity)</b>	Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.
<b>refframe</b> → <b>load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.
<b>refframe</b> → <b>more3DCalibration()</b>	Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.
<b>refframe</b> → <b>nextRefFrame()</b>	Continue l'énumération des référentiels commencée à l'aide de yFirstRefFrame ( ).
<b>refframe</b> → <b>registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>refframe</b> → <b>save3DCalibration()</b>	Applique les paramètres de calibration tridimensionnelle précédemment calculés.
<b>refframe</b> → <b>set_bearing(newval)</b>	

Modifie le cap de référence utilisé par le compas.

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

Modifie le nom logique du référentiel.

**refframe**→**set\_mountPosition**(**position**, **orientation**)

Modifie le référentiel de la boussole et des inclinomètres.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

**refframe**→**start3DCalibration**()

Initie le processus de calibration tridimensionnelle des capteurs.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YRefFrame.FindRefFrame()****YRefFrame****yFindRefFrame()YRefFrame.FindRefFrame()**

Permet de retrouver un référentiel d'après un identifiant donné.

```
def FindRefFrame( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRefFrame.isOnline()` pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le référentiel sans ambiguïté

**Retourne :**

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

## **YRefFrame.FirstRefFrame() yFirstRefFrame()YRefFrame.FirstRefFrame()**

**YRefFrame**

Commence l'énumération des référentiels accessibles par la librairie.

```
def FirstRefFrame( )
```

Utiliser la fonction `YRefFrame.nextRefFrame( )` pour itérer sur les autres référentiels.

**Retourne :**

un pointeur sur un objet `YRefFrame`, correspondant au premier référentiel accessible en ligne, ou `null` si il n'y a pas de référentiels disponibles.

**refframe→cancel3DCalibration()**  
**refframe.cancel3DCalibration()**

---

**YRefFrame**

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

```
def cancel3DCalibration( )
```

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**refframe→describe()refframe.describe()****YRefFrame**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le référentiel (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**refframe→get\_3DCalibrationHint()**

**YRefFrame**

**refframe→3DCalibrationHint()**

**refframe.get\_3DCalibrationHint()**

---

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

def **get\_3DCalibrationHint( )**

**Retourne :**

une chaîne de caractères.

---

**refframe→get\_3DCalibrationLogMsg()**  
**refframe→3DCalibrationLogMsg()**  
**refframe.get\_3DCalibrationLogMsg()**

---

**YRefFrame**

Retourne le dernier message de log produit par le processus de calibration.

```
def get_3DCalibrationLogMsg( )
```

Si aucun nouveau message n'est disponible, retourne une chaîne vide.

**Retourne :**

une chaîne de caractères.

**refframe→get\_3DCalibrationProgress()**

**YRefFrame**

**refframe→3DCalibrationProgress()**

**refframe.get\_3DCalibrationProgress()**

---

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.

```
def get_3DCalibrationProgress( )
```

**Retourne :**

une nombre entier entre 0 (pas commencé) et 100 (terminé).

---

**refframe→get\_3DCalibrationStage()****YRefFrame****refframe→3DCalibrationStage()****refframe.get\_3DCalibrationStage()**

---

Retourne l'index de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

```
def get_3DCalibrationStage( )
```

**Retourne :**

une nombre entier, croissant au fur et à mesure de la complétion des étapes.

**refframe→get\_3DCalibrationStageProgress()**

**YRefFrame**

**refframe→3DCalibrationStageProgress()**

**refframe.get\_3DCalibrationStageProgress()**

---

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

**def get\_3DCalibrationStageProgress( )**

**Retourne :**

une nombre entier entre 0 (pas commencé) et 100 (terminé).

---

**refframe→get\_advertisedValue()****YRefFrame****refframe→advertisedValue()****refframe.get\_advertisedValue()**

---

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du référentiel (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**refframe**→**get\_bearing()**

**YRefFrame**

**refframe**→**bearing()****refframe.get\_bearing()**

---

Retourne le cap de référence utilisé par le compas.

```
def get_bearing( )
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici.

**Retourne :**

une valeur numérique représentant le cap de référence utilisé par le compas

En cas d'erreur, déclenche une exception ou retourne `Y_BEARING_INVALID`.



---

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

---

**YRefFrame**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

**refframe→get\_errorType()**

**YRefFrame**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

---

**refframe**→**get\_friendlyName()****YRefFrame****refframe**→**friendlyName()****refframe.get\_friendlyName()**

---

Retourne un identifiant global du référentiel au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du référentiel si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du référentiel (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le référentiel en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YRefFrame**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**refframe**→**get\_functionId()****YRefFrame****refframe**→**functionId()****refframe.get\_functionId()**

---

Retourne l'identifiant matériel du référentiel, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**refframe**→**get\_hardwareId()**

**YRefFrame**

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

---

Retourne l'identifiant matériel unique du référentiel au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du référentiel (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**refframe**→**get\_logicalName()****YRefFrame****refframe**→**logicalName()****refframe.get\_logicalName()**

---

Retourne le nom logique du référentiel.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du référentiel. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**refframe**→**get\_module()**

**YRefFrame**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`



**refframe→get\_mountOrientation()****YRefFrame****refframe→mountOrientation()****refframe.get\_mountOrientation()**

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

```
def get_mountOrientation( )
```

**Retourne :**

une valeur parmi l'énumération `Y_MOUNTORIENTATION` (`Y_MOUNTORIENTATION_TWELVE`, `Y_MOUNTORIENTATION_THREE`, `Y_MOUNTORIENTATION_SIX`, `Y_MOUNTORIENTATION_NINE`) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face `BOTTOM` le 12h pointe vers l'avant, tandis que sur la face `TOP` le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get\_mountPosition()**

**YRefFrame**

**refframe→mountPosition()**

**refframe.get\_mountPosition()**

---

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

```
def get_mountPosition( )
```

**Retourne :**

une valeur parmi l'énumération Y\_MOUNTPOSITION (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), correspondant à l'installation dans une boîte, sur l'une des six faces

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**refframe→get\_userdata()****YRefFrame****refframe→userdata()refframe.get\_userdata()**

---

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## refframe→isOnline()refframe.isOnline()

YRefFrame

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du référentiel sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le référentiel est joignable, `false` sinon

**refframe→load()refframe.load()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→more3DCalibration()**  
**refframe.more3DCalibration()**

**YRefFrame**

---

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

```
def more3DCalibration( )
```

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode `get_3DCalibrationHint` (les instructions changent pendant la procédure de calibration). En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**refframe**→**nextRefFrame()****refframe.nextRefFrame()****YRefFrame**

---

Continue l'énumération des référentiels commencée à l'aide de `yFirstRefFrame()`.

```
def nextRefFrame( )
```

**Retourne :**

un pointeur sur un objet `YRefFrame` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**refframe→registerValueCallback()**  
**refframe.registerValueCallback()****YRefFrame**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.



---

**refframe→save3DCalibration()**  
**refframe.save3DCalibration()**

---

**YRefFrame**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

```
def save3DCalibration( )
```

N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé après le redémarrage du module. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe**→**set\_bearing()****YRefFrame****refframe**→**setBearing()****refframe.set\_bearing()**

Modifie le cap de référence utilisé par le compas.

```
def set_bearing( newval)
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici. Par exemple, si vous indiquez comme cap de référence la valeur de la déclinaison magnétique terrestre, le compas donnera l'orientation par rapport au Nord géographique. De même, si le capteur n'est pas positionné dans une des directions standard à cause d'un angle de lacet supplémentaire, vous pouvez le configurer comme cap de référence afin que le compas donne la direction naturelle attendue.

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur numérique représentant le cap de référence utilisé par le compas

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**refframe→set\_logicalName()****YRefFrame****refframe→setLogicalName()****refframe.set\_logicalName()**

---

Modifie le nom logique du référentiel.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du référentiel.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_mountPosition()****YRefFrame****refframe→setMountPosition()****refframe.set\_mountPosition()**

Modifie le référentiel de la boussole et des inclinomètres.

```
def set_mountPosition( position, orientation)
```

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas où le module n'est pas utilisé horizontalement et à l'endroit, il faut indiquer son orientation de référence (parallèle à la surface terrestre) afin que les mesures soient faites relativement à cette position.

**Paramètres :**

**position** une valeur parmi l'énumération Y\_MOUNTPOSITION (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), correspondant à l'installation dans une boîte, sur l'une des six faces.

**orientation** une valeur parmi l'énumération Y\_MOUNTORIENTATION (Y\_MOUNTORIENTATION\_TWELVE, Y\_MOUNTORIENTATION\_THREE, Y\_MOUNTORIENTATION\_SIX, Y\_MOUNTORIENTATION\_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière. N'oubliez pas d'appeler la méthode saveToFlash() du module si le réglage doit être préservé.

---

**refframe**→**set\_userdata()****YRefFrame****refframe**→**setUserData()****refframe.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**refframe→start3DCalibration()**  
**refframe.start3DCalibration()**

---

**YRefFrame**

Initie le processus de calibration tridimensionnelle des capteurs.

**def start3DCalibration( )**

Cette calibration est utilisée à bas niveau pour l'estimation inertielle de position et pour améliorer la précision des mesures d'inclinaison. Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.35. Interface de la fonction Relay

La librairie de programmation Yoctopuce permet simplement de changer l'état du relais. Le changement d'état n'est pas persistant: le relais retournera spontanément à sa position de repos dès que le module est mis hors tension ou redémarré. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Pour les modules dotés de deux sorties par relais (relai inverseur), les deux sorties sont appelées A et B, la sortie A correspondant à la position de repos (hors tension) et la sortie B correspondant à l'état actif. Si vous préféreriez l'état par défaut opposé, vous pouvez simplement changer vos fils sur le bornier.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_relay.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YRelay = yoctolib.YRelay;
php	require_once('yocto_relay.php');
c++	#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 *

### Fonction globales

#### yFindRelay(func)

Permet de retrouver un relais d'après un identifiant donné.

#### yFirstRelay()

Commence l'énumération des relais accessibles par la librairie.

### Méthodes des objets YRelay

#### relay→delayedPulse(ms\_delay, ms\_duration)

Pré-programme une impulsion

#### relay→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### relay→get\_advertisedValue()

Retourne la valeur courante du relais (pas plus de 6 caractères).

#### relay→get\_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

#### relay→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

#### relay→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

#### relay→get\_friendlyName()

Retourne un identifiant global du relais au format NOM\_MODULE . NOM\_FONCTION.

#### relay→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### relay→get\_functionId()

Retourne l'identifiant matériel du relais, sans référence au module.

**relay→get\_hardwareId()**

Retourne l'identifiant matériel unique du relais au format `SERIAL . FUNCTIONID`.

**relay→get\_logicalName()**

Retourne le nom logique du relais.

**relay→get\_maxTimeOnStateA()**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**relay→get\_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

**relay→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**relay→get\_output()**

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

**relay→get\_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

**relay→get\_state()**

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

**relay→get\_stateAtPowerOn()**

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, `UNCHANGED` pour aucun changement).

**relay→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**relay→isOnline()**

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

**relay→load(msValidity)**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

**relay→nextRelay()**

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

**relay→pulse(ms\_duration)**

Commute le relais à l'état B (actif) pour une durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

**relay→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**relay→set\_logicalName(newval)**

Modifie le nom logique du relais.

**relay→set\_maxTimeOnStateA(newval)**

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**relay→set\_maxTimeOnStateB(newval)**



Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

**relay→set\_output(newval)**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

**relay→set\_state(newval)**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

**relay→set\_stateAtPowerOn(newval)**

Pré-programme l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

**relay→set\_userdata(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YRelay.FindRelay()****YRelay****yFindRelay()YRelay.FindRelay()**

Permet de retrouver un relais d'après un identifiant donné.

```
def FindRelay( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le relais soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRelay.isOnline()` pour tester si le relais est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le relais sans ambiguïté

**Retourne :**

un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

## YRelay.FirstRelay() yFirstRelay()YRelay.FirstRelay()

YRelay

Commence l'énumération des relais accessibles par la librairie.

```
def FirstRelay( )
```

Utiliser la fonction `YRelay.nextRelay( )` pour itérer sur les autres relais.

**Retourne :**

un pointeur sur un objet `YRelay`, correspondant au premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

**relay**→**delayedPulse()****relay.delayedPulse()**

---

**YRelay**

Pré-programme une impulsion

```
def delayedPulse( ms_delay, ms_duration)
```

**Paramètres :**

**ms\_delay**     délai d'attente avant l'impulsion, en millisecondes

**ms\_duration**     durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→describe()relay.describe()****YRelay**

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le relais (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**relay**→**get\_advertisedValue()**

**YRelay**

**relay**→**advertisedValue()****relay.get\_advertisedValue()**

---

Retourne la valeur courante du relais (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du relais (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**relay→get\_countdown()****YRelay****relay→countdown()relay.get\_countdown()**

---

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

```
def get_countdown( )
```

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

**relay**→**get\_errorMessage()**

**YRelay**

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du relais.



---

**relay**→**get\_errorType()****YRelay****relay**→**errorType()****relay.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du relais.

**relay**→**get\_friendlyName()**

**YRelay**

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

---

Retourne un identifiant global du relais au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du relais si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du relais (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le relais en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

---

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

---

**YRelay**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**relay**→**get\_functionId()**

**YRelay**

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

---

Retourne l'identifiant matériel du relais, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le relais (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

**relay**→**get\_hardwareId()****YRelay****relay**→**hardwareId()****relay.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du relais au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du relais (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le relais (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**relay**→**get\_logicalName()**

**YRelay**

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

---

Retourne le nom logique du relais.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du relais. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**relay**→**get\_maxTimeOnStateA()****YRelay****relay**→**maxTimeOnStateA()****relay.get\_maxTimeOnStateA()**

---

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
def get_maxTimeOnStateA( )
```

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEA_INVALID`.

**relay**→**get\_maxTimeOnStateB()**

**YRelay**

**relay**→**maxTimeOnStateB()**

**relay.get\_maxTimeOnStateB()**

---

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
def get_maxTimeOnStateB( )
```

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEB\_INVALID.



**relay**→**get\_module()****YRelay****relay**→**module()****relay.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**relay**→**get\_output()**

**YRelay**

**relay**→**output()****relay.get\_output()**

---

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
def get_output( )
```

**Retourne :**

soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUT\_INVALID.

---

**relay**→**get\_pulseTimer()****YRelay****relay**→**pulseTimer()****relay.get\_pulseTimer()**

---

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
def get_pulseTimer( )
```

Si aucune impulsion n'est en cours, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne Y\_PULSETIMER\_INVALID.

**relay**→**get\_state()**

**YRelay**

**relay**→**state()****relay.get\_state()**

---

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

```
def get_state( )
```

**Retourne :**

soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y\_STATE\_INVALID.

---

**relay**→**get\_stateAtPowerOn()****YRelay****relay**→**stateAtPowerOn()****relay.get\_stateAtPowerOn()**

---

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
def get_stateAtPowerOn( )
```

**Retourne :**

une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B représentant l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y\_STATEATPOWERON\_INVALID.

**relay**→**get\_userData()**

**YRelay**

**relay**→**userData()****relay.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**relay→isOnline()relay.isOnline()****YRelay**

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le relais est joignable, `false` sinon

**relay→load()relay.load()****YRelay**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**relay**→**nextRelay()****relay.nextRelay()****YRelay**

---

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

```
def nextRelay( )
```

**Retourne :**

un pointeur sur un objet `YRelay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

## relay→pulse()relay.pulse()

YRelay

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
def pulse( ms_duration)
```

### Paramètres :

**ms\_duration** durée de l'impulsion, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**relay→registerValueCallback()**  
**relay.registerValueCallback()**

---

**YRelay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**relay**→**set\_logicalName()**

**YRelay**

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

---

Modifie le nom logique du relais.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du relais.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**relay**→**set\_maxTimeOnStateA()****YRelay****relay**→**setMaxTimeOnStateA()****relay.set\_maxTimeOnStateA()**

---

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
def set_maxTimeOnStateA( newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_maxTimeOnStateB()**

**YRelay**

**relay→setMaxTimeOnStateB()**

**relay.set\_maxTimeOnStateB()**

---

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
def set_maxTimeOnStateB( newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay**→**set\_output()****YRelay****relay**→**setOutput()****relay.set\_output()**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
def set_output( newval)
```

**Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay**→**set\_state()**

**YRelay**

**relay**→**setState()****relay.set\_state()**

---

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
def set_state( newval)
```

**Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**relay→set\_stateAtPowerOn()**  
**relay→setStateAtPowerOn()**  
**relay.set\_stateAtPowerOn()**

**YRelay**

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
def set_stateAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay**→**set\_userdata()**

**YRelay**

**relay**→**setUserData()****relay.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.36. Interface des fonctions de type senseur

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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 *

### Fonction globales

#### yFindSensor(func)

Permet de retrouver un senseur d'après un identifiant donné.

#### yFirstSensor()

Commence l'énumération des senseurs accessibles par la librairie.

### Méthodes des objets YSensor

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### sensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### sensor→get\_advertisedValue()

Retourne la valeur courante du senseur (pas plus de 6 caractères).

#### sensor→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### sensor→get\_currentValue()

Retourne la valeur actuelle de la mesure.

#### sensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

#### sensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

#### sensor→get\_friendlyName()

Retourne un identifiant global du senseur au format `NOM_MODULE . NOM_FONCTION`.

#### sensor→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### sensor→get\_functionId()

Retourne l'identifiant matériel du senseur, sans référence au module.

#### sensor→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur au format `SERIAL.FUNCTIONID`.

#### **sensor→get\_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

#### **sensor→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

#### **sensor→get\_logicalName()**

Retourne le nom logique du capteur.

#### **sensor→get\_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

#### **sensor→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **sensor→get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **sensor→get\_recordedData(startTime, endTime)**

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

#### **sensor→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

#### **sensor→get\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **sensor→get\_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

#### **sensor→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **sensor→isOnline()**

Vérifie si le module hébergeant le capteur est joignable, sans déclencher d'erreur.

#### **sensor→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur est joignable, sans déclencher d'erreur.

#### **sensor→load(msValidity)**

Met en cache les valeurs courantes du capteur, avec une durée de validité spécifiée.

#### **sensor→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

#### **sensor→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur, avec une durée de validité spécifiée.

#### **sensor→nextSensor()**

Continue l'énumération des capteurs commencée à l'aide de `yFirstSensor()`.

#### **sensor→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### **sensor→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

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

Modifie la mémoire de valeur maximale observée.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du senseur.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**sensor**→**set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YSensor.FindSensor()****YSensor****yFindSensor()YSensor.FindSensor()**

Permet de retrouver un senseur d'après un identifiant donné.

```
def FindSensor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le senseur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSensor.isOnline()` pour tester si le senseur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le senseur sans ambiguïté

**Retourne :**

un objet de classe `YSensor` qui permet ensuite de contrôler le senseur.

---

**YSensor.FirstSensor()  
yFirstSensor()YSensor.FirstSensor()**

---

**YSensor**

Commence l'énumération des senseurs accessibles par la librairie.

```
def FirstSensor( )
```

Utiliser la fonction `YSensor.nextSensor( )` pour itérer sur les autres senseurs.

**Retourne :**

un pointeur sur un objet `YSensor`, correspondant au premier senseur accessible en ligne, ou `null` si il n'y a pas de senseurs disponibles.

**sensor→calibrateFromPoints()**  
**sensor.calibrateFromPoints()****YSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**sensor→describe()****sensor.describe()****YSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le senseur (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**sensor**→**get\_advertisedValue()**

**YSensor**

**sensor**→**advertisedValue()**

**sensor.get\_advertisedValue()**

---

Retourne la valeur courante du senseur (pas plus de 6 caractères).

def **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante du senseur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**sensor**→**get\_currentRawValue()****YSensor****sensor**→**currentRawValue()****sensor.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**sensor**→**get\_currentValue()**

**YSensor**

**sensor**→**currentValue()****sensor.get\_currentValue()**

---

Retourne la valeur actuelle de la mesure.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la mesure

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**sensor**→**get\_errorMessage()****YSensor****sensor**→**errorMessage()****sensor.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la bibliothèque Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur.

**sensor**→**get\_errorType()**

**YSensor**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

---

**sensor**→**get\_friendlyName()****YSensor****sensor**→**friendlyName()****sensor.get\_friendlyName()**

---

Retourne un identifiant global du senseur au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du senseur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du senseur (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le senseur en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**sensor**→**get\_functionDescriptor()**

**YSensor**

**sensor**→**functionDescriptor()**

**sensor.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID



---

**sensor**→**get\_functionId()****YSensor****sensor**→**functionId()****sensor.get\_functionId()**

---

Retourne l'identifiant matériel du senseur, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**sensor**→**get\_hardwareId()**

**YSensor**

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

---

Retourne l'identifiant matériel unique du senseur au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du senseur (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**sensor**→**get\_highestValue()****YSensor****sensor**→**highestValue()****sensor.get\_highestValue()**

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

**sensor**→**get\_logFrequency()**

**YSensor**

**sensor**→**logFrequency()****sensor.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def **get\_logFrequency()** ( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

**sensor**→**get\_logicalName()****YSensor****sensor**→**logicalName()****sensor.get\_logicalName()**

---

Retourne le nom logique du senseur.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du senseur. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**sensor**→**get\_lowestValue()**

**YSensor**

**sensor**→**lowestValue()****sensor.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

---

**sensor**→**get\_module()****YSensor****sensor**→**module()****sensor.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**sensor**→**get\_recordedData()****YSensor****sensor**→**recordedData()****sensor.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.



---

**sensor→get\_reportFrequency()****YSensor****sensor→reportFrequency()****sensor.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**sensor**→**get\_resolution()**

**YSensor**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

**sensor**→**get\_unit()****YSensor****sensor**→**unit()****sensor.get\_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**sensor**→**get\_userData()**

**YSensor**

**sensor**→**userData()****sensor.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**sensor**→**isOnline()****sensor.isOnline()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du senseur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le senseur est joignable, `false` sinon

**sensor**→**load()****sensor.load()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**sensor→loadCalibrationPoints()**  
**sensor.loadCalibrationPoints()**

---

**YSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor**→**nextSensor()****sensor.nextSensor()**

**YSensor**

---

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

def **nextSensor**( )

**Retourne :**

un pointeur sur un objet `YSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.



---

**sensor→registerTimedReportCallback()**  
**sensor.registerTimedReportCallback()**

---

**YSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**sensor**→**registerValueCallback()****YSensor****sensor.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**sensor**→**set\_highestValue()****YSensor****sensor**→**setHighestValue()****sensor.set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor**→**set\_logFrequency()****YSensor****sensor**→**setLogFrequency()****sensor.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor**→**set\_logicalName()****YSensor****sensor**→**setLogicalName()****sensor.set\_logicalName()**

Modifie le nom logique du senseur.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du senseur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor**→**set\_lowestValue()**

**YSensor**

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

---

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor**→**set\_reportFrequency()**  
**sensor**→**setReportFrequency()**  
**sensor.set\_reportFrequency()**

**YSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor**→**set\_resolution()**

**YSensor**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**sensor**→**set\_userdata()****YSensor****sensor**→**setUserData()****sensor.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.37. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'intervalle de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindServo(func)

Permet de retrouver un servo d'après un identifiant donné.

#### yFirstServo()

Commence l'énumération des servo accessibles par la librairie.

### Méthodes des objets YServo

#### servo→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### servo→get\_advertisedValue()

Retourne la valeur courante du servo (pas plus de 6 caractères).

#### servo→get\_enabled()

Retourne l'état de fonctionnement du \$FUNCTION\$.

#### servo→get\_enabledAtPowerOn()

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

#### servo→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

#### servo→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

#### servo→get\_friendlyName()

Retourne un identifiant global du servo au format `NOM_MODULE . NOM_FONCTION`.

#### servo→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### servo→get\_functionId()

Retourne l'identifiant matériel du servo, sans référence au module.

#### servo→get\_hardwareId()

Retourne l'identifiant matériel unique du servo au format `SERIAL . FUNCTIONID`.

#### servo→get\_logicalName()

Retourne le nom logique du servo.

**servo→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**servo→get\_neutral()**

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

**servo→get\_position()**

Retourne la position courante du servo.

**servo→get\_positionAtPowerOn()**

Retourne la position du servo au démarrage du module.

**servo→get\_range()**

Retourne la plage d'utilisation du servo.

**servo→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**servo→isOnline()**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

**servo→load(msValidity)**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

**servo→move(target, ms\_duration)**

Déclenche un mouvement à vitesse constante vers une position donnée.

**servo→nextServo()**

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

**servo→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**servo→set\_enabled(newval)**

Démarre ou arrête le \$FUNCTION\$.

**servo→set\_enabledAtPowerOn(newval)**

Configure l'état du générateur de signal de commande du servo au démarrage du module.

**servo→set\_logicalName(newval)**

Modifie le nom logique du servo.

**servo→set\_neutral(newval)**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

**servo→set\_position(newval)**

Modifie immédiatement la consigne de position du servo.

**servo→set\_positionAtPowerOn(newval)**

Configure la position du servo au démarrage du module.

**servo→set\_range(newval)**

Modifie la plage d'utilisation du servo, en pourcents.

**servo→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

### 3. Reference

---

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YServo.FindServo() yFindServo()YServo.FindServo()

## YServo

Permet de retrouver un servo d'après un identifiant donné.

```
def FindServo( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le servo soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le servo sans ambiguïté

### Retourne :

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

## **YServo.FirstServo() yFirstServo()YServo.FirstServo()**

---

**YServo**

Commence l'énumération des servo accessibles par la librairie.

```
def FirstServo( )
```

Utiliser la fonction `YServo.nextServo( )` pour itérer sur les autres servo.

### **Retourne :**

un pointeur sur un objet `YServo`, correspondant au premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

**servo→describe()****servo.describe()****YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le servo (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**servo**→**get\_advertisedValue()**

**YServo**

**servo**→**advertisedValue()****servo.get\_advertisedValue()**

---

Retourne la valeur courante du servo (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du servo (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.



---

**servo→get\_enabled()****YServo****servo→enabled()****servo.get\_enabled()**

---

Retourne l'état de fonctionnement du \$FUNCTION\$.

```
def get_enabled( )
```

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du \$FUNCTION\$

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**servo→get\_enabledAtPowerOn()**

**YServo**

**servo→enabledAtPowerOn()**

**servo.get\_enabledAtPowerOn()**

---

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

```
def get_enabledAtPowerOn( )
```

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

---

**servo→get\_errorMessage()****YServo****servo→errorMessage()servo.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du servo.

**servo→get\_errorType()**

**YServo**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du servo.

---

**servo→get\_friendlyName()****YServo****servo→friendlyName()****servo.get\_friendlyName()**

---

Retourne un identifiant global du servo au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du servo si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du servo (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le servo en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YServo**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

def **get\_functionDescriptor()** ( )

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**servo**→**get\_functionId()****YServo****servo**→**functionId()****servo.get\_functionId()**

---

Retourne l'identifiant matériel du servo, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le servo (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**servo**→**get\_hardwareId()**

**YServo**

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

---

Retourne l'identifiant matériel unique du servo au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du servo (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le servo (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.



---

**servo→get\_logicalName()****YServo****servo→logicalName()****servo.get\_logicalName()**

---

Retourne le nom logique du servo.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du servo. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**servo**→**get\_module()**

**YServo**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

**servo→get\_neutral()****YServo****servo→neutral()servo.get\_neutral()**

---

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

```
def get_neutral( )
```

**Retourne :**

un entier représentant la durée en microsecondes de l'impulsion correspondant au neutre du servo

En cas d'erreur, déclenche une exception ou retourne Y\_NEUTRAL\_INVALID.

**servo→get\_position()**

**YServo**

**servo→position()****servo.get\_position()**

---

Retourne la position courante du servo.

```
def get_position( )
```

**Retourne :**

un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne Y\_POSITION\_INVALID.

---

**servo→get\_positionAtPowerOn()****YServo****servo→positionAtPowerOn()****servo.get\_positionAtPowerOn()**

---

Retourne la position du servo au démarrage du module.

```
def get_positionAtPowerOn( )
```

**Retourne :**

un entier représentant la position du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_POSITIONATPOWERON\_INVALID.

**servo**→**get\_range()**

**YServo**

**servo**→**range()****servo.get\_range()**

---

Retourne la plage d'utilisation du servo.

```
def get_range( )
```

**Retourne :**

un entier représentant la plage d'utilisation du servo

En cas d'erreur, déclenche une exception ou retourne `Y_RANGE_INVALID`.

---

**servo→get\_userdata()****YServo****servo→userData()servo.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **servo→isOnline()****servo.isOnline()**

**YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache du servo sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le servo est joignable, `false` sinon



**servo→load()****servo.load()****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→move()****servo.move()****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

```
def move( target, ms_duration)
```

**Paramètres :**

**target** nouvelle position à la fin du mouvement  
**ms\_duration** durée totale du mouvement, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**servo**→**nextServo()****servo.nextServo()****YServo**

---

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

```
def nextServo( )
```

**Retourne :**

un pointeur sur un objet `YServo` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**servo→registerValueCallback()  
servo.registerValueCallback()****YServo**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**servo**→**set\_enabled()****YServo****servo**→**setEnabled()****servo.set\_enabled()**

Démarre ou arrête le \$FUNCTION\$.

```
def set_enabled( newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo**→**set\_enabledAtPowerOn()**

**YServo**

**servo**→**setEnabledAtPowerOn()**

**servo.set\_enabledAtPowerOn()**

---

Configure l'état du générateur de signal de commande du servo au démarrage du module.

```
def set_enabledAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**servo**→**set\_logicalName()****YServo****servo**→**setLogicalName()****servo.set\_logicalName()**

---

Modifie le nom logique du servo.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du servo.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo**→**set\_neutral()**

**YServo**

**servo**→**setNeutral()****servo.set\_neutral()**

---

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
def set_neutral( newval)
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**servo**→**set\_position()****YServo****servo**→**setPosition()****servo.set\_position()**

Modifie immédiatement la consigne de position du servo.

```
def set_position( newval)
```

**Paramètres :**

**newval** un entier représentant immédiatement la consigne de position du servo

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo**→**set\_positionAtPowerOn()**

**YServo**

**servo**→**setPositionAtPowerOn()**

**servo.set\_positionAtPowerOn()**

---

Configure la position du servo au démarrage du module.

```
def set_positionAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_range()****YServo****servo→setRange()****servo.set\_range()**

Modifie la plage d'utilisation du servo, en pourcents.

```
def set_range( newval)
```

La valeur 100% correspond à un signal de commande standard, variant de 1 [ms] à 2 [ms]. Pour les servos supportent une plage double, de 0.5 [ms] à 2.5 [ms], vous pouvez utiliser une valeur allant jusqu'à 200%. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la plage d'utilisation du servo, en pourcents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo**→**set\_userdata()**

**YServo**

**servo**→**setUserData()****servo.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.38. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindTemperature(func)

Permet de retrouver un capteur de température d'après un identifiant donné.

#### yFirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

### Méthodes des objets YTemperature

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### temperature→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### temperature→get\_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

#### temperature→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### temperature→get\_currentValue()

Retourne la valeur actuelle de la température.

#### temperature→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

#### temperature→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

#### temperature→get\_friendlyName()

Retourne un identifiant global du capteur de température au format `NOM_MODULE . NOM_FONCTION`.

#### temperature→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### temperature→get\_functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

#### temperature→get\_hardwareId()

	Retourne l'identifiant matériel unique du capteur de température au format SERIAL . FUNCTIONID.
<b>temperature</b> → <b>get_highestValue()</b>	Retourne la valeur maximale observée pour la température depuis le démarrage du module.
<b>temperature</b> → <b>get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>temperature</b> → <b>get_logicalName()</b>	Retourne le nom logique du capteur de température.
<b>temperature</b> → <b>get_lowestValue()</b>	Retourne la valeur minimale observée pour la température depuis le démarrage du module.
<b>temperature</b> → <b>get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>temperature</b> → <b>get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>temperature</b> → <b>get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>temperature</b> → <b>get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>temperature</b> → <b>get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>temperature</b> → <b>get_sensorType()</b>	Retourne le type de capteur de température utilisé par le module
<b>temperature</b> → <b>get_unit()</b>	Retourne l'unité dans laquelle la température est exprimée.
<b>temperature</b> → <b>get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>temperature</b> → <b>isOnline()</b>	Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.
<b>temperature</b> → <b>isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.
<b>temperature</b> → <b>load(msValidity)</b>	Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.
<b>temperature</b> → <b>loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>temperature</b> → <b>load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.
<b>temperature</b> → <b>nextTemperature()</b>	Continue l'énumération des capteurs de température commencée à l'aide de yFirstTemperature ( ).
<b>temperature</b> → <b>registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>temperature</b> → <b>registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>temperature</b> → <b>set_highestValue(newval)</b>	

Modifie la mémoire de valeur maximale observée.

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

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur de température.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**temperature**→**set\_sensorType**(newval)

Change le type de senseur utilisé par le module.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YTemperature.FindTemperature()****YTemperature****yFindTemperature()YTemperature.FindTemperature()**

Permet de retrouver un capteur de température d'après un identifiant donné.

```
def FindTemperature( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de température soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de température sans ambiguïté

**Retourne :**

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.



**YTemperature.FirstTemperature()****YTemperature****yFirstTemperature()YTemperature.FirstTemperature()**

Commence l'énumération des capteurs de température accessibles par la librairie.

```
def FirstTemperature( )
```

Utiliser la fonction `YTemperature.nextTemperature( )` pour itérer sur les autres capteurs de température.

**Retourne :**

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

**temperature→calibrateFromPoints()  
temperature.calibrateFromPoints()****YTemperature**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**temperature→describe()temperature.describe()****YTemperature**

---

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de température (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**temperature→get\_advertisedValue()**

**YTemperature**

**temperature→advertisedValue()**

**temperature.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

def **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**temperature**→**get\_currentRawValue()****YTemperature****temperature**→**currentRawValue()****temperature.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**temperature**→**get\_currentValue()**

**YTemperature**

**temperature**→**currentValue()**

**temperature.get\_currentValue()**

---

Retourne la valeur actuelle de la température.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la température

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**temperature**→**get\_errorMessage()****YTemperature****temperature**→**errorMessage()****temperature.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

**temperature→get\_errorType()**

**YTemperature**

**temperature→errorType()**

**temperature.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.



**temperature→get\_friendlyName()****YTemperature****temperature→friendlyName()****temperature.get\_friendlyName()**

---

Retourne un identifiant global du capteur de température au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de température si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de température (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de température en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**temperature→get\_functionDescriptor()**

**YTemperature**

**temperature→functionDescriptor()**

**temperature.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

**YTemperature**

---

Retourne l'identifiant matériel du capteur de température, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de température (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**temperature→get\_hardwareId()**

**YTemperature**

**temperature→hardwareId()**

**temperature.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de température (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**temperature**→**get\_highestValue()****YTemperature****temperature**→**highestValue()****temperature.get\_highestValue()**

---

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

**temperature→get\_logFrequency()**

**YTemperature**

**temperature→logFrequency()**

**temperature.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**def get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

**temperature**→**get\_logicalName()****YTemperature****temperature**→**logicalName()****temperature.get\_logicalName()**

---

Retourne le nom logique du capteur de température.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de température. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**temperature→get\_lowestValue()**

**YTemperature**

**temperature→lowestValue()**

**temperature.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.



**temperature**→**get\_module()****YTemperature****temperature**→**module()****temperature.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**temperature→get\_recordedData()**

**YTemperature**

**temperature→recordedData()**

**temperature.get\_recordedData()**

---

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**temperature**→**get\_reportFrequency()****YTemperature****temperature**→**reportFrequency()****temperature.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**temperature**→**get\_resolution()**

**YTemperature**

**temperature**→**resolution()**

**temperature.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**temperature**→**get\_sensorType()****YTemperature****temperature**→**sensorType()****temperature.get\_sensorType()**

---

Retourne le type de capteur de température utilisé par le module

```
def get_sensorType( )
```

**Retourne :**

une valeur parmi 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 et Y\_SENSORTYPE\_PT100\_2WIRES représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_SENSORTYPE\_INVALID.

**temperature**→**get\_unit()**

**YTemperature**

**temperature**→**unit()****temperature.get\_unit()**

---

Retourne l'unité dans laquelle la température est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

---

**temperature**→**get\_userdata()****YTemperature****temperature**→**userData()****temperature.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**temperature**→**isOnline()****temperature.isOnline()**

**YTemperature**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de température est joignable, `false` sinon



**temperature→load()temperature.load()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→loadCalibrationPoints()**  
**temperature.loadCalibrationPoints()**

**YTemperature**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**temperature**→**nextTemperature()**  
**temperature.nextTemperature()**

---

**YTemperature**

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

def **nextTemperature()**

**Retourne :**

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**temperature→registerTimedReportCallback()  
temperature.registerTimedReportCallback()****YTemperature**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

**temperature**→**registerValueCallback()**  
**temperature.registerValueCallback()**

---

**YTemperature**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**temperature→set\_highestValue()**

**YTemperature**

**temperature→setHighestValue()**

**temperature.set\_highestValue()**

---

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YTemperature**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logicalName()**

**YTemperature**

**temperature→setLogicalName()**

**temperature.set\_logicalName()**

---

Modifie le nom logique du capteur de température.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de température.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



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

**YTemperature**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_reportFrequency()**

**YTemperature**

**temperature→setReportFrequency()**

**temperature.set\_reportFrequency()**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YTemperature**

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_sensorType()****YTemperature****temperature→setSensorType()****temperature.set\_sensorType()**

Change le type de senseur utilisé par le module.

```
def set_sensorType( newval)
```

Cette fonction sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi 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 et Y\_SENSORTYPE\_PT100\_2WIRES

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature**→**set\_userdata()****YTemperature****temperature**→**setUserData()****temperature.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.39. Interface de la fonction Tilt

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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');
c++	#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 *

### Fonction globales

#### yFindTilt(func)

Permet de retrouver un inclinomètre d'après un identifiant donné.

#### yFirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

### Méthodes des objets YTilt

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### tilt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### tilt→get\_advertisedValue()

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

#### tilt→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### tilt→get\_currentValue()

Retourne la valeur actuelle de l'inclinaison.

#### tilt→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

#### tilt→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

#### tilt→get\_friendlyName()

Retourne un identifiant global de l'inclinomètre au format `NOM_MODULE . NOM_FONCTION`.

#### tilt→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### tilt→get\_functionId()

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

#### tilt→get\_hardwareId()

Retourne l'identifiant matériel unique de l'inclinomètre au format `SERIAL . FUNCTIONID`.

**tilt→get\_highestValue()**

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

**tilt→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**tilt→get\_logicalName()**

Retourne le nom logique de l'inclinomètre.

**tilt→get\_lowestValue()**

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

**tilt→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**tilt→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**tilt→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**tilt→get\_unit()**

Retourne l'unité dans laquelle l'inclinaison est exprimée.

**tilt→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**tilt→isOnline()**

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

**tilt→load(msValidity)**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

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

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

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

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

**tilt→nextTilt()**

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

**tilt→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**tilt→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**tilt→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

**tilt→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

### 3. Reference

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

Modifie le nom logique de l'inclinomètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **tilt→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **tilt→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.



## YTilt.FindTilt() yFindTilt()YTilt.FindTilt()

YTilt

Permet de retrouver un inclinomètre d'après un identifiant donné.

```
def FindTilt( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'inclinomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTilt.isOnline()` pour tester si l'inclinomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'inclinomètre sans ambiguïté

### Retourne :

un objet de classe `YTilt` qui permet ensuite de contrôler l'inclinomètre.

**YTilt.FirstTilt()**

**YTilt**

**yFirstTilt()YTilt.FirstTilt()**

---

Commence l'énumération des inclinomètres accessibles par la librairie.

```
def FirstTilt( )
```

Utiliser la fonction `YTilt.nextTilt()` pour itérer sur les autres inclinomètres.

**Retourne :**

un pointeur sur un objet `YTilt`, correspondant au premier inclinomètre accessible en ligne, ou `null` si il n'y a pas de inclinomètres disponibles.

**tilt→calibrateFromPoints()tilt.calibrateFromPoints()****YTilt**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→describe()tilt.describe()****YTilt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

**def describe( )**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant l'inclinomètre (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**tilt**→**get\_advertisedValue()****YTilt****tilt**→**advertisedValue()****tilt.get\_advertisedValue()**

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**tilt**→**get\_currentRawValue()**

**YTilt**

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

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**tilt**→**get\_currentValue()****YTilt****tilt**→**currentValue()****tilt.get\_currentValue()**

Retourne la valeur actuelle de l'inclinaison.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'inclinaison

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**tilt**→**get\_errorMessage()**

**YTilt**

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.



**tilt**→**get\_errorType()****YTilt****tilt**→**errorType()****tilt.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

**tilt**→**get\_friendlyName()****YTilt****tilt**→**friendlyName()****tilt.get\_friendlyName()**

Retourne un identifiant global de l'inclinomètre au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'inclinomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'inclinomètre (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**tilt**→**get\_functionDescriptor()****YTilt****tilt**→**functionDescriptor()****tilt.get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**tilt**→**get\_functionId()**

**YTilt**

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

---

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**tilt**→**get\_hardwareId()****YTilt****tilt**→**hardwareId()****tilt.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'inclinomètre au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'inclinomètre (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**tilt**→**get\_highestValue()**

**YTilt**

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

---

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

**tilt**→**get\_logFrequency()****YTilt****tilt**→**logFrequency()****tilt.get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**tilt**→**get\_logicalName()**

**YTilt**

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

---

Retourne le nom logique de l'inclinomètre.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'inclinomètre. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.



**tilt**→**get\_lowestValue()****YTilt****tilt**→**lowestValue()****tilt.get\_lowestValue()**

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**tilt**→**get\_module()**

**YTilt**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**tilt**→**get\_recordedData()****YTilt****tilt**→**recordedData()****tilt.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**tilt→get\_reportFrequency()**

**YTilt**

**tilt→reportFrequency()****tilt.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

def **get\_reportFrequency()**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**tilt**→**get\_resolution()****YTilt****tilt**→**resolution()****tilt.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**tilt**→**get\_unit()**

**YTilt**

**tilt**→**unit()****tilt.get\_unit()**

---

Retourne l'unité dans laquelle l'inclinaison est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'inclinaison est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**tilt**→**get\_userData()****YTilt****tilt**→**userData()****tilt.get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

**def get\_userData( )**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache de l'inclinomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'inclinomètre est joignable, false sinon



**tilt→load()tilt.load()****YTilt**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→loadCalibrationPoints()  
tilt.loadCalibrationPoints()****YTilt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**tilt**→**nextTilt()****tilt.nextTilt()****YTilt**

---

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

```
def nextTilt( )
```

**Retourne :**

un pointeur sur un objet `YTilt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**tilt→registerTimedReportCallback()**  
**tilt.registerTimedReportCallback()**

YTilt

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

**tilt→registerValueCallback()**  
**tilt.registerValueCallback()**

---

**YTilt**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**tilt**→**set\_highestValue()**

**YTilt**

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

---

Modifie la mémoire de valeur maximale observée.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt**→**set\_logFrequency()****YTilt****tilt**→**setLogFrequency()****tilt.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt**→**set\_logicalName()****YTilt****tilt**→**setLogicalName()****tilt.set\_logicalName()**

Modifie le nom logique de l'inclinomètre.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'inclinomètre.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**tilt**→**set\_lowestValue()****YTilt****tilt**→**setLowestValue()****tilt.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_reportFrequency()****YTilt****tilt→setReportFrequency()tilt.set\_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt**→**set\_resolution()****YTilt****tilt**→**setResolution()****tilt.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt**→**set\_userdata()**

**YTilt**

**tilt**→**setUserData()****tilt.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.40. Interface de la fonction Voc

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindVoc(func)

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

#### yFirstVoc()

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

### Méthodes des objets YVoc

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### voc→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### voc→get\_advertisedValue()

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

#### voc→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### voc→get\_currentValue()

Retourne la mesure actuelle du taux de VOC estimé.

#### voc→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

#### voc→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

#### voc→get\_friendlyName()

Retourne un identifiant global du capteur de Composés Organiques Volatils au format NOM\_MODULE . NOM\_FONCTION.

#### voc→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### voc→get\_functionId()

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

### 3. Reference

#### **voc**→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.

#### **voc**→get\_highestValue()

Retourne la valeur maximale observée pour le taux de VOC estimé.

#### **voc**→get\_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

#### **voc**→get\_logicalName()

Retourne le nom logique du capteur de Composés Organiques Volatils.

#### **voc**→get\_lowestValue()

Retourne la valeur minimale observée pour le taux de VOC estimé.

#### **voc**→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **voc**→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **voc**→get\_recordedData(startTime, endTime)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

#### **voc**→get\_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

#### **voc**→get\_resolution()

Retourne la résolution des valeurs mesurées.

#### **voc**→get\_unit()

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

#### **voc**→get\_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

#### **voc**→isOnline()

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

#### **voc**→isOnline\_async(callback, context)

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

#### **voc**→load(msValidity)

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

#### **voc**→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **voc**→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

#### **voc**→nextVoc()

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de yFirstVoc().

#### **voc**→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**voc→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**voc→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

**voc→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**voc→set\_logicalName(newval)**

Modifie le nom logique du capteur de Composés Organiques Volatils.

**voc→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

**voc→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**voc→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**voc→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userData.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YVoc.FindVoc()****YVoc****yFindVoc()YVoc.FindVoc()**

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

```
def FindVoc( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de Composés Organiques Volatils soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoc.isOnline()` pour tester si le capteur de Composés Organiques Volatils est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de Composés Organiques Volatils sans ambiguïté

**Retourne :**

un objet de classe `YVoc` qui permet ensuite de contrôler le capteur de Composés Organiques Volatils.



## **YVoc.FirstVoc()** **yFirstVoc()YVoc.FirstVoc()**

**YVoc**

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

```
def FirstVoc( )
```

Utiliser la fonction `YVoc.nextVoc()` pour itérer sur les autres capteurs de Composés Organiques Volatils.

**Retourne :**

un pointeur sur un objet `YVoc`, correspondant au premier capteur de Composés Organiques Volatils accessible en ligne, ou `null` si il n'y a pas de capteurs de Composés Organiques Volatils disponibles.

**voc→calibrateFromPoints()****voc.calibrateFromPoints()****YVoc**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→describe()****voc.describe()****YVoc**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**voc**→**get\_advertisedValue()**

**YVoc**

**voc**→**advertisedValue()****voc.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**voc**→**get\_currentRawValue()****YVoc****voc**→**currentRawValue()****voc.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

**voc**→**get\_currentValue()**

**YVoc**

**voc**→**currentValue()****voc.get\_currentValue()**

---

Retourne la mesure actuelle du taux de VOC estimé.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la mesure actuelle du taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

**voc→get\_errorMessage()****YVoc****voc→errorMessage()voc.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

**voc**→**get\_errorType()**

**YVoc**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.



**voc**→**get\_friendlyName()****YVoc****voc**→**friendlyName()****voc.get\_friendlyName()**

Retourne un identifiant global du capteur de Composés Organiques Volatils au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de Composés Organiques Volatils si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**voc**→**get\_functionDescriptor()**

**YVoc**

**voc**→**functionDescriptor()**

**voc.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voc**→**get\_functionId()****YVoc****voc**→**functionId()****voc.get\_functionId()**

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voc**→**get\_hardwareId()**

**YVoc**

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

---

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format `SERIAL.FUNCTIONID`.

def **get\_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**voc**→**get\_highestValue()****YVoc****voc**→**highestValue()****voc.get\_highestValue()**

Retourne la valeur maximale observée pour le taux de VOC estimé.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**voc**→**get\_logFrequency()**

**YVoc**

**voc**→**logFrequency()****voc.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

def **get\_logFrequency()** ( )

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**voc**→**get\_logicalName()****YVoc****voc**→**logicalName()****voc.get\_logicalName()**

Retourne le nom logique du capteur de Composés Organiques Volatils.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**voc**→**get\_lowestValue()**

**YVoc**

**voc**→**lowestValue()****voc.get\_lowestValue()**

---

Retourne la valeur minimale observée pour le taux de VOC estimé.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.



**voc**→**get\_module()****YVoc****voc**→**module()****voc.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**voc**→**get\_recordedData()****YVoc****voc**→**recordedData()****voc.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voc**→**get\_reportFrequency()****YVoc****voc**→**reportFrequency()****voc.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**voc**→**get\_resolution()**

**YVoc**

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

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**voc**→**get\_unit()****YVoc****voc**→**unit()****voc.get\_unit()**

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le taux de VOC estimé est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**voc**→**get\_userData()**

**YVoc**

**voc**→**userData()****voc.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**voc→isOnline()****voc.isOnline()****YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de Composés Organiques Volatils sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de Composés Organiques Volatils est joignable, `false` sinon

**voc→load()****voc.load()****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**voc→loadCalibrationPoints()**  
**voc.loadCalibrationPoints()**

---

**YVoc**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc**→**nextVoc()****voc.nextVoc()**

**YVoc**

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

def **nextVoc()**

**Retourne :**

un pointeur sur un objet `YVoc` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**voc→registerTimedReportCallback()**  
**voc.registerTimedReportCallback()**

---

**YVoc**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

**voc**→**registerValueCallback()****YVoc****voc.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**voc**→**set\_highestValue()****YVoc****voc**→**setHighestValue()****voc.set\_highestValue()**

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour le taux de VOC estimé

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc**→**set\_logFrequency()****YVoc****voc**→**setLogFrequency()****voc.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc**→**set\_logicalName()****YVoc****voc**→**setLogicalName()****voc.set\_logicalName()**

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc**→**set\_lowestValue()**

**YVoc**

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

---

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour le taux de VOC estimé

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**voc**→**set\_reportFrequency()****YVoc****voc**→**setReportFrequency()****voc.set\_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc**→**set\_resolution()**

**YVoc**

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

---

Modifie la résolution des valeurs physique mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc**→**set\_userdata()****YVoc****voc**→**setUserData()****voc.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.41. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindVoltage(func)

Permet de retrouver un capteur de tension d'après un identifiant donné.

#### yFirstVoltage()

Commence l'énumération des capteurs de tension accessibles par la librairie.

### Méthodes des objets YVoltage

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

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### voltage→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE ( NAME ) = SERIAL . FUNCTIONID`.

#### voltage→get\_advertisedValue()

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

#### voltage→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### voltage→get\_currentValue()

Retourne la valeur instantanée de la tension.

#### voltage→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### voltage→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### voltage→get\_friendlyName()

Retourne un identifiant global du capteur de tension au format `NOM_MODULE . NOM_FONCTION`.

#### voltage→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### voltage→get\_functionId()

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

#### voltage→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de tension au format `SERIAL . FUNCTIONID`.

**`voltage→get_highestValue()`**

Retourne la valeur maximale observée pour la tension.

**`voltage→get_logFrequency()`**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**`voltage→get_logicalName()`**

Retourne le nom logique du capteur de tension.

**`voltage→get_lowestValue()`**

Retourne la valeur minimale observée pour la tension.

**`voltage→get_module()`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`voltage→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`voltage→get_recordedData(startTime, endTime)`**

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

**`voltage→get_reportFrequency()`**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**`voltage→get_resolution()`**

Retourne la résolution des valeurs mesurées.

**`voltage→get_unit()`**

Retourne l'unité dans laquelle la tension est exprimée.

**`voltage→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**`voltage→isOnline()`**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

**`voltage→isOnline_async(callback, context)`**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

**`voltage→load(msValidity)`**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

**`voltage→loadCalibrationPoints(rawValues, refValues)`**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

**`voltage→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

**`voltage→nextVoltage()`**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

**`voltage→registerTimedReportCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**`voltage→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`voltage→set_highestValue(newval)`**

Modifie la mémoire de valeur maximale observée pour la tension.

**`voltage→set_logFrequency(newval)`**

### 3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

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

Modifie le nom logique du capteur de tension.

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

Modifie la mémoire de valeur minimale observée pour la tension.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs mesurées.

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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YVoltage.FindVoltage() yFindVoltage()YVoltage.FindVoltage()

## YVoltage

Permet de retrouver un capteur de tension d'après un identifiant donné.

```
def FindVoltage( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltage.isOnline()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de tension sans ambiguïté

### Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

**YVoltage.FirstVoltage()**

**YVoltage**

**yFirstVoltage()****YVoltage.FirstVoltage()**

---

Commence l'énumération des capteurs de tension accessibles par la librairie.

```
def FirstVoltage( )
```

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

**Retourne :**

un pointeur sur un objet `YVoltage`, correspondant au premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.



**voltage→calibrateFromPoints()**  
**voltage.calibrateFromPoints()****YVoltage**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
def calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter [support@yoctopuce.com](mailto:support@yoctopuce.com).

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→describe()****voltage.describe()****YVoltage**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le capteur de tension (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

**voltage**→**get\_advertisedValue()****YVoltage****voltage**→**advertisedValue()****voltage.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**voltage**→**get\_currentRawValue()**

**YVoltage**

**voltage**→**currentRawValue()**

**voltage.get\_currentRawValue()**

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
def get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

---

**voltage**→**get\_currentValue()****YVoltage****voltage**→**currentValue()****voltage.get\_currentValue()**

---

Retourne la valeur instantanée de la tension.

```
def get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur instantanée de la tension

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**voltage**→**get\_errorMessage()**

**YVoltage**

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

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

---

**voltage**→**get\_errorType()****YVoltage****voltage**→**errorType()****voltage.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

**voltage→get\_friendlyName()**

**YVoltage**

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

---

Retourne un identifiant global du capteur de tension au format `NOM_MODULE . NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de tension (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le capteur de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.



---

**voltage**→**get\_functionDescriptor()****YVoltage****voltage**→**functionDescriptor()****voltage.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voltage→get\_functionId()**

**YVoltage**

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

---

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voltage**→**get\_hardwareId()****YVoltage****voltage**→**hardwareId()****voltage.get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de tension au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de tension (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**voltage**→**get\_highestValue()**

**YVoltage**

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

---

Retourne la valeur maximale observée pour la tension.

```
def get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la tension

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

---

**voltage**→**get\_logFrequency()****YVoltage****voltage**→**logFrequency()****voltage.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
def get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**voltage→get\_logicalName()**

**YVoltage**

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

---

Retourne le nom logique du capteur de tension.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de tension. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**voltage**→**get\_lowestValue()****YVoltage****voltage**→**lowestValue()****voltage.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la tension.

```
def get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la tension

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**voltage**→**get\_module()**

**YVoltage**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`



**voltage**→**get\_recordedData()****YVoltage****voltage**→**recordedData()****voltage.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
def get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voltage→get\_reportFrequency()**

**YVoltage**

**voltage→reportFrequency()**

**voltage.get\_reportFrequency()**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
def get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

**voltage**→**get\_resolution()****YVoltage****voltage**→**resolution()****voltage.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
def get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

**voltage**→**get\_unit()**

**YVoltage**

**voltage**→**unit()****voltage.get\_unit()**

---

Retourne l'unité dans laquelle la tension est exprimée.

```
def get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

---

**voltage**→**get\_userdata()****YVoltage****voltage**→**userData()****voltage.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **voltage→isOnline()voltage.isOnline()**

**YVoltage**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de tension est joignable, `false` sinon

**voltage→load()****voltage.load()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→loadCalibrationPoints()**

**YVoltage**

**voltage.loadCalibrationPoints()**

---

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
def loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**voltage**→**nextVoltage()****voltage.nextVoltage()****YVoltage**

---

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

```
def nextVoltage( )
```

**Retourne :**

un pointeur sur un objet `YVoltage` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**voltage→registerTimedReportCallback()**  
**voltage.registerTimedReportCallback()****YVoltage**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

**voltage→registerValueCallback()**  
**voltage.registerValueCallback()**

---

**YVoltage**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**voltage**→**set\_highestValue()**

**YVoltage**

**voltage**→**setHighestValue()**

**voltage.set\_highestValue()**

---

Modifie la mémoire de valeur maximale observée pour la tension.

```
def set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la tension

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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

**YVoltage**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
def set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage**→**set\_logicalName()**

**YVoltage**

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

---

Modifie le nom logique du capteur de tension.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de tension.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage**→**set\_lowestValue()****YVoltage****voltage**→**setLowestValue()****voltage.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée pour la tension.

```
def set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la tension

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_reportFrequency()**

**YVoltage**

**voltage→setReportFrequency()**

**voltage.set\_reportFrequency()**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

```
def set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**voltage**→**set\_resolution()****YVoltage****voltage**→**setResolution()****voltage.set\_resolution()**

---

Modifie la résolution des valeurs mesurées.

```
def set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage**→**set\_userdata()**

**YVoltage**

**voltage**→**setUserData()****voltage.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.42. Interface de la fonction Source de tension

La librairie de programmation Yoctopuce permet de commander la tension de sortie du module. Vous pouvez affecter une valeur fixe, ou faire des transitions de voltage.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
<b>yFindVSource(func)</b> Permet de retrouver une source de tension d'après un identifiant donné.
<b>yFirstVSource()</b> Commence l'énumération des sources de tension accessibles par la librairie.
Méthodes des objets YVSource
<b>vsource→describe()</b> Retourne un court texte décrivant la fonction au format <code>TYPE ( NAME ) = SERIAL . FUNCTIONID</code> .
<b>vsource→get_advertisedValue()</b> Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
<b>vsource→get_errorMessage()</b> Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>vsource→get_errorType()</b> Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>vsource→get_extPowerFailure()</b> Rend TRUE si le voltage de l'alimentation externe est trop bas.
<b>vsource→get_failure()</b> Indique si le module est en condition d'erreur.
<b>vsource→get_friendlyName()</b> Retourne un identifiant global de la fonction au format <code>NOM_MODULE . NOM_FONCTION</code> .
<b>vsource→get_functionDescriptor()</b> Retourne un identifiant unique de type <code>YFUN_DESCRIPTOR</code> correspondant à la fonction.
<b>vsource→get_functionId()</b> Retourne l'identifiant matériel de la fonction, sans référence au module.
<b>vsource→get_hardwareId()</b> Retourne l'identifiant matériel unique de la fonction au format <code>SERIAL . FUNCTIONID</code> .
<b>vsource→get_logicalName()</b> Retourne le nom logique de la source de tension.
<b>vsource→get_module()</b> Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>vsource→get_module_async(callback, context)</b>

### 3. Reference

	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b><code>vsource→get_overCurrent()</code></b>	Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.
<b><code>vsource→get_overHeat()</code></b>	Rend TRUE si le module est en surchauffe.
<b><code>vsource→get_overLoad()</code></b>	Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.
<b><code>vsource→get_regulationFailure()</code></b>	Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.
<b><code>vsource→get_unit()</code></b>	Retourne l'unité dans laquelle la tension est exprimée.
<b><code>vsource→get_userData()</code></b>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<b><code>vsource→get_voltage()</code></b>	Retourne la valeur de la commande de tension de sortie en mV
<b><code>vsource→isOnline()</code></b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b><code>vsource→isOnline_async(callback, context)</code></b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b><code>vsource→load(msValidity)</code></b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b><code>vsource→load_async(msValidity, callback, context)</code></b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b><code>vsource→nextVSource()</code></b>	Continue l'énumération des sources de tension commencée à l'aide de <code>yFirstVSource()</code> .
<b><code>vsource→pulse(voltage, ms_duration)</code></b>	Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.
<b><code>vsource→registerValueCallback(callback)</code></b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b><code>vsource→set_logicalName(newval)</code></b>	Modifie le nom logique de la source de tension.
<b><code>vsource→set_userData(data)</code></b>	Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
<b><code>vsource→set_voltage(newval)</code></b>	Règle la tension de sortie du module (en milliVolts).
<b><code>vsource→voltageMove(target, ms_duration)</code></b>	Déclenche une variation constante de la sortie vers une valeur donnée.
<b><code>vsource→wait_async(callback, context)</code></b>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**yFindVSource()** —**YVSource****YVSource.FindVSource()****YVSource.FindVSource()**

Permet de retrouver une source de tension d'après un identifiant donné.

def **FindVSource**( **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

**Retourne :**

un objet de classe `YVSource` qui permet ensuite de contrôler la source de tension.

**yFirstVSource()** —

**YVSource**

**YVSource.FirstVSource()****YVSource.FirstVSource()**

---

Commence l'énumération des sources de tension accessibles par la librairie.

def **FirstVSource()**

Utiliser la fonction `YVSource.nextVSource()` pour itérer sur les autres sources de tension.

**Retourne :**

un pointeur sur un objet `YVSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de sources de tension disponibles.

**vsource**→**get\_advertisedValue()****YVSource****vsource**→**advertisedValue()****vsource.get\_advertisedValue()**


---

 Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
 

---

```
def get_advertisedValue( )
```

**vsource**→**get\_advertisedValue()****vsource**→**advertisedValue()****vsource.get\_advertisedValue()**


---

 Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
 

---

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

**Retourne :**

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères)

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**vsourc**→**get\_errorMessage()**

**YVSource**

**vsourc**→**errorMessage()**

**vsourc.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

def **get\_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la fonction



---

**vsources**→**getErrorType()****YVSource****vsources**→**errorType()****vsources.getErrorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

**def getErrorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

**vsourc**→**get\_extPowerFailure()****YVSource****vsourc**→**extPowerFailure()****vsourc.get\_extPowerFailure()**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

```
def get_extPowerFailure( )
```

**vsourc**→**get\_extPowerFailure()****vsourc**→**extPowerFailure()****vsourc.get\_extPowerFailure()**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

js	function <b>get_extPowerFailure</b> ( )
php	function <b>get_extPowerFailure</b> ( )
cpp	Y_EXTPOWERFAILURE_enum <b>get_extPowerFailure</b> ( )
m	-(Y_EXTPOWERFAILURE_enum) extPowerFailure
pas	function <b>get_extPowerFailure</b> ( ): Integer
vb	function <b>get_extPowerFailure</b> ( ) As Integer
cs	int <b>get_extPowerFailure</b> ( )
java	int <b>get_extPowerFailure</b> ( )
py	def <b>get_extPowerFailure</b> ( )
cmd	YVSource <b>target</b> <b>get_extPowerFailure</b>

**Retourne :**

soit Y\_EXTPOWERFAILURE\_FALSE, soit Y\_EXTPOWERFAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_EXTPOWERFAILURE\_INVALID.

**vsource**→**get\_failure()****YVSource****vsource**→**failure()****vsource.get\_failure()**

Indique si le module est en condition d'erreur.

**def get\_failure( )****vsource**→**get\_failure()****vsource**→**failure()****vsource.get\_failure()**

Indique si le module est en condition d'erreur.

js	<b>function get_failure( )</b>
php	<b>function get_failure( )</b>
cpp	<b>Y_FAILURE_enum get_failure( )</b>
m	<b>-(Y_FAILURE_enum) failure</b>
pas	<b>function get_failure( ): Integer</b>
vb	<b>function get_failure( ) As Integer</b>
cs	<b>int get_failure( )</b>
java	<b>int get_failure( )</b>
py	<b>def get_failure( )</b>
cmd	<b>YVSource target get_failure</b>

Il possible de savoir de quelle erreur il s'agit en testant get\_overheat, get\_overcurrent etc... Lorsqu'un condition d'erreur est rencontrée, la tension de sortie est mise à zéro est ne peut pas être changée tant la fonction reset() n'aura pas appelée.

**Retourne :**

soit Y\_FAILURE\_FALSE, soit Y\_FAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_FAILURE\_INVALID.

**vsourc**→**get\_functionDescriptor()**

**YVSource**

**vsourc**→**functionDescriptor()**

**vsourc.get\_vsourcDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

def **get\_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**vsource**→**get\_logicalName()****YVSource****vsource**→**logicalName()****vsource.get\_logicalName()**


---

 Retourne le nom logique de la source de tension.
 

---

```
def get_logicalName( )
```

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


---

 Retourne le nom logique de la source de tension.
 

---

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

**Retourne :**

une chaîne de caractères représentant le nom logique de la source de tension

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**vsources**→**get\_module()**

**YVSource**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`def get_module( )`

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**vsource**→**get\_overCurrent()****YVSource****vsource**→**overCurrent()****vsource.get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
def get_overCurrent( )
```

**vsource**→**get\_overCurrent()****vsource**→**overCurrent()****vsource.get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

js	function <b>get_overCurrent</b> ( )
php	function <b>get_overCurrent</b> ( )
cpp	Y_OVERCURRENT_enum <b>get_overCurrent</b> ( )
m	-(Y_OVERCURRENT_enum) overCurrent
pas	function <b>get_overCurrent</b> ( ): Integer
vb	function <b>get_overCurrent</b> ( ) As Integer
cs	int <b>get_overCurrent</b> ( )
java	int <b>get_overCurrent</b> ( )
py	def <b>get_overCurrent</b> ( )
cmd	YVSource <b>target</b> <b>get_overCurrent</b>

**Retourne :**

soit Y\_OVERCURRENT\_FALSE, soit Y\_OVERCURRENT\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERCURRENT\_INVALID.

**vsourc**→**get\_overHeat()****YVSource****vsourc**→**overHeat()****vsourc.get\_overHeat()**

Rend TRUE si le module est en surchauffe.

```
def get_overHeat( )
```

**vsourc**→**get\_overHeat()****vsourc**→**overHeat()****vsourc.get\_overHeat()**

Rend TRUE si le module est en surchauffe.

js	function <b>get_overHeat</b> ( )
php	function <b>get_overHeat</b> ( )
cpp	Y_OVERHEAT_enum <b>get_overHeat</b> ( )
m	-(Y_OVERHEAT_enum) overHeat
pas	function <b>get_overHeat</b> ( ): Integer
vb	function <b>get_overHeat</b> ( ) As Integer
cs	int <b>get_overHeat</b> ( )
java	int <b>get_overHeat</b> ( )
py	def <b>get_overHeat</b> ( )
cmd	YVSource <b>target</b> <b>get_overHeat</b>

**Retourne :**

soit Y\_OVERHEAT\_FALSE, soit Y\_OVERHEAT\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERHEAT\_INVALID.



**vsource**→**get\_overLoad()****YVSource****vsource**→**overLoad()****vsource.get\_overLoad()**


---

 Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.
 

---

```
def get_overLoad( )
```

**vsource**→**get\_overLoad()****vsource**→**overLoad()****vsource.get\_overLoad()**


---

 Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.
 

---

js	function <b>get_overLoad</b> ( )
php	function <b>get_overLoad</b> ( )
cpp	Y_OVERLOAD_enum <b>get_overLoad</b> ( )
m	-(Y_OVERLOAD_enum) overLoad
pas	function <b>get_overLoad</b> ( ): Integer
vb	function <b>get_overLoad</b> ( ) As Integer
cs	int <b>get_overLoad</b> ( )
java	int <b>get_overLoad</b> ( )
py	def <b>get_overLoad</b> ( )
cmd	YVSource <b>target</b> <b>get_overLoad</b>

**Retourne :**

soit Y\_OVERLOAD\_FALSE, soit Y\_OVERLOAD\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERLOAD\_INVALID.

**vsourc**→**get\_regulationFailure()****YVSource****vsourc**→**regulationFailure()****vsourc.get\_regulationFailure()**


---

 Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.
 

---

```
def get_regulationFailure( )
```

**vsourc**→**get\_regulationFailure()****vsourc**→**regulationFailure()****vsourc.get\_regulationFailure()**


---

 Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.
 

---

js	function <b>get_regulationFailure</b> ( )
php	function <b>get_regulationFailure</b> ( )
cpp	Y_REGULATIONFAILURE_enum <b>get_regulationFailure</b> ( )
m	-(Y_REGULATIONFAILURE_enum) regulationFailure
pas	function <b>get_regulationFailure</b> ( ): Integer
vb	function <b>get_regulationFailure</b> ( ) As Integer
cs	int <b>get_regulationFailure</b> ( )
java	int <b>get_regulationFailure</b> ( )
py	def <b>get_regulationFailure</b> ( )
cmd	YVSource <b>target</b> <b>get_regulationFailure</b>

**Retourne :**

soit Y\_REGULATIONFAILURE\_FALSE, soit Y\_REGULATIONFAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_REGULATIONFAILURE\_INVALID.

**vsource**→**get\_unit()****YVSource****vsource**→**unit()****vsource.get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
def get_unit( )
```

**vsource**→**get\_unit()****vsource**→**unit()****vsource.get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

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

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**vsource**→**get\_userData()**

**YVSource**

**vsource**→**userData()****vsource.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

def **get\_userData()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

**vsource**→**get\_voltage()****YVSource****vsource**→**voltage()****vsource.get\_voltage()**

---

Retourne la valeur de la commande de tension de sortie en mV

def **get\_voltage**( )

**Retourne :**

un entier représentant la valeur de la commande de tension de sortie en mV

En cas d'erreur, déclenche une exception ou retourne Y\_VOLTAGE\_INVALID.

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

def **isOnline**( )

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si la fonction est joignable, `false` sinon

**vsource→load()****vsource.load()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

def **load**( **msValidity**)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsources**→**nextVSource()****vsources.nextVSource()**

**YVSource**

---

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

def **nextVSource()**

**Retourne :**

un pointeur sur un objet `YVSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.



**vsource→pulse()****vsource.pulse()****YVSource**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
def pulse( voltage, ms_duration)
```

**vsource→pulse()****vsource.pulse()**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

js	function pulse( voltage, ms_duration)
php	function pulse( \$voltage, \$ms_duration)
cpp	int pulse( int voltage, int ms_duration)
m	-(int) pulse : (int) voltage : (int) ms_duration
pas	function pulse( voltage: integer, ms_duration: integer): integer
vb	function pulse( ByVal voltage As Integer, ByVal ms_duration As Integer) As Integer
cs	int pulse( int voltage, int ms_duration)
java	int pulse( int voltage, int ms_duration)
py	def pulse( voltage, ms_duration)
cmd	YVSource target pulse voltage ms_duration

**Paramètres :**

**voltage** tension demandée, en millivolts  
**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsouce**→**registerValueCallback()****YVSource****vsouce.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

**vsouce**→**registerValueCallback()****vsouce.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function <b>registerValueCallback</b> ( callback )
php	function <b>registerValueCallback</b> ( \$callback )
cpp	void <b>registerValueCallback</b> ( YDisplayUpdateCallback callback )
pas	procedure <b>registerValueCallback</b> ( callback: TGenericUpdateCallback )
vb	procedure <b>registerValueCallback</b> ( ByVal callback As GenericUpdateCallback )
cs	void <b>registerValueCallback</b> ( UpdateCallback callback )
java	void <b>registerValueCallback</b> ( UpdateCallback callback )
py	def <b>registerValueCallback</b> ( callback )
m	-(void) <b>registerValueCallback</b> : (YFunctionUpdateCallback) callback

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**vsource**→**set\_logicalName()****YVSource****vsource**→**setLogicalName()****vsource.set\_logicalName()**


---

 Modifie le nom logique de la source de tension.
 

---

```
def set_logicalName( newval)
```

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


---

 Modifie le nom logique de la source de tension.
 

---

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

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la source de tension

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsourceset\_userdata()**

**YVSource**

**vsourcesetUserData()vsourceset\_userdata()**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

def **set\_userdata( data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**vsourceset\_voltage()****YVSource****vsourcesetVoltage()vsourceset\_voltage()**

Règle la tension de sortie du module (en milliVolts).

def **set\_voltage**( **newval**)**vsourceset\_voltage()****vsourcesetVoltage()vsourceset\_voltage()**

Règle la tension de sortie du module (en milliVolts).

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

**Paramètres :****newval** un entier**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsource→voltageMove()vsource.voltageMove()****YVSource**

Déclenche une variation constante de la sortie vers une valeur donnée.

```
def voltageMove( target, ms_duration)
```

**vsource→voltageMove()vsource.voltageMove()**

Déclenche une variation constante de la sortie vers une valeur donnée.

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

**Paramètres :**

**target** nouvelle valeur de sortie à la fin de la transition, en milliVolts.  
**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.43. Interface de la fonction WakeUpMonitor

La fonction WakeUpMonitor prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindWakeUpMonitor(func)

Permet de retrouver un moniteur d'après un identifiant donné.

#### yFirstWakeUpMonitor()

Commence l'énumération des Moniteurs accessibles par la librairie.

### Méthodes des objets YWakeUpMonitor

#### wakeupmonitor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

#### wakeupmonitor→get\_advertisedValue()

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

#### wakeupmonitor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

#### wakeupmonitor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

#### wakeupmonitor→get\_friendlyName()

Retourne un identifiant global du moniteur au format `NOM_MODULE . NOM_FONCTION`.

#### wakeupmonitor→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### wakeupmonitor→get\_functionId()

Retourne l'identifiant matériel du moniteur, sans référence au module.

#### wakeupmonitor→get\_hardwareId()

Retourne l'identifiant matériel unique du moniteur au format `SERIAL . FUNCTIONID`.

#### wakeupmonitor→get\_logicalName()

Retourne le nom logique du moniteur.

#### wakeupmonitor→get\_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupmonitor→get\_module\_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**wakeupmonitor→get\_nextWakeUp()**

Retourne la prochaine date/heure de réveil agendée (format UNIX)

**wakeupmonitor→get\_powerDuration()**

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

**wakeupmonitor→get\_sleepCountdown()**

Retourne le temps avant le prochain sommeil.

**wakeupmonitor→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

**wakeupmonitor→get\_wakeUpReason()**

Renvoie la raison du dernier réveil.

**wakeupmonitor→get\_wakeUpState()**

Revoie l'état actuel du moniteur

**wakeupmonitor→isOnline()**

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

**wakeupmonitor→load(msValidity)**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

**wakeupmonitor→nextWakeUpMonitor()**

Continue l'énumération des Moniteurs commencée à l'aide de yFirstWakeUpMonitor().

**wakeupmonitor→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**wakeupmonitor→resetSleepCountDown()**

Réinitialise le compteur de mise en sommeil.

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

Modifie le nom logique du moniteur.

**wakeupmonitor→set\_nextWakeUp(newval)**

Modifie les jours de la semaine où un réveil doit avoir lieu.

**wakeupmonitor→set\_powerDuration(newval)**

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

**wakeupmonitor→set\_sleepCountdown(newval)**

Modifie le temps avant le prochain sommeil .

**wakeupmonitor→set\_userdata(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

**wakeupmonitor→sleep(secBeforeSleep)**

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

**wakeupmonitor→sleepFor(secUntilWakeUp, secBeforeSleep)**

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

**wakeupmonitor→sleepUntil(wakeUpTime, secBeforeSleep)**

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

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



Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**wakeupmonitor**→**wakeUp()**

Force un réveil.

## **YWakeUpMonitor.FindWakeUpMonitor() yFindWakeUpMonitor() YWakeUpMonitor.FindWakeUpMonitor()**

**YWakeUpMonitor**

Permet de retrouver un moniteur d'après un identifiant donné.

```
def FindWakeUpMonitor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.IsOnline()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### **Paramètres :**

**func** une chaîne de caractères qui référence le moniteur sans ambiguïté

### **Retourne :**

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

**YWakeUpMonitor.FirstWakeUpMonitor()**  
**yFirstWakeUpMonitor()**  
**YWakeUpMonitor.FirstWakeUpMonitor()**

**YWakeUpMonitor**

---

Commence l'énumération des Moniteurs accessibles par la librairie.

def **FirstWakeUpMonitor()**

Utiliser la fonction `YWakeUpMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

**Retourne :**

un pointeur sur un objet `YWakeUpMonitor`, correspondant au premier moniteur accessible en ligne, ou `null` si il n'y a pas de Moniteurs disponibles.

**wakeupmonitor→describe()**  
**wakeupmonitor.describe()****YWakeUpMonitor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

def **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le moniteur (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

---

**wakeupmonitor**→**get\_advertisedValue()****YWakeUpMonitor****wakeupmonitor**→**advertisedValue()****wakeupmonitor.get\_advertisedValue()**

---

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wakeupmonitor→get\_errorMessage()**

**YWakeUpMonitor**

**wakeupmonitor→errorMessage()**

**wakeupmonitor.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

---

**wakeupmonitor→get\_errorType()****YWakeUpMonitor****wakeupmonitor→errorType()****wakeupmonitor.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

**wakeupmonitor→get\_friendlyName()**

**YWakeUpMonitor**

**wakeupmonitor→friendlyName()**

**wakeupmonitor.get\_friendlyName()**

---

Retourne un identifiant global du moniteur au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du moniteur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du moniteur (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le moniteur en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.



---

**wakeupmonitor**→**get\_functionDescriptor()****YWakeUpMonitor****wakeupmonitor**→**functionDescriptor()****wakeupmonitor.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**wakeupmonitor→get\_functionId()**

**YWakeUpMonitor**

**wakeupmonitor→functionId()**

**wakeupmonitor.get\_functionId()**

---

Retourne l'identifiant matériel du moniteur, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le moniteur (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

**wakeupmonitor→get\_hardwareId()****YWakeUpMonitor****wakeupmonitor→hardwareId()****wakeupmonitor.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du moniteur au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moniteur (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le moniteur (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

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

---

**YWakeUpMonitor**

Retourne le nom logique du moniteur.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du moniteur. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**wakeupmonitor→get\_module()****YWakeUpMonitor****wakeupmonitor→module()****wakeupmonitor.get\_module()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**wakeupmonitor→get\_nextWakeUp()**

**YWakeUpMonitor**

**wakeupmonitor→nextWakeUp()**

**wakeupmonitor.get\_nextWakeUp()**

---

Retourne la prochaine date/heure de réveil agendée (format UNIX)

```
def get_nextWakeUp( )
```

**Retourne :**

un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTWAKEUP\_INVALID.

---

**wakeupmonitor→get\_powerDuration()****YWakeUpMonitor****wakeupmonitor→powerDuration()****wakeupmonitor.get\_powerDuration()**

---

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
def get_powerDuration( )
```

**Retourne :**

un entier représentant le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement

En cas d'erreur, déclenche une exception ou retourne Y\_POWERDURATION\_INVALID.

wakeupmonitor→get\_sleepCountdown()

YWakeUpMonitor

wakeupmonitor→sleepCountdown()

wakeupmonitor.get\_sleepCountdown()

---

Retourne le temps avant le prochain sommeil.

def get\_sleepCountdown( )

**Retourne :**

un entier représentant le temps avant le prochain sommeil

En cas d'erreur, déclenche une exception ou retourne Y\_SLEEP\_COUNTDOWN\_INVALID.



**wakeupmonitor→get\_userdata()****YWakeUpMonitor****wakeupmonitor→userData()****wakeupmonitor.get\_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userdata.

```
def get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**wakeupmonitor→get\_wakeUpReason()**

**YWakeUpMonitor**

**wakeupmonitor→wakeUpReason()**

**wakeupmonitor.get\_wakeUpReason()**

Renvoie la raison du dernier réveil.

**def get\_wakeUpReason( )**

**Retourne :**

une valeur parmi 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 et  
Y\_WAKEUPREASON\_SCHEDULE6

En cas d'erreur, déclenche une exception ou retourne Y\_WAKEUPREASON\_INVALID.

---

**wakeupmonitor**→**get\_wakeUpState()****YWakeUpMonitor****wakeupmonitor**→**wakeUpState()****wakeupmonitor.get\_wakeUpState()**

---

Revoie l'état actuel du moniteur

```
def get_wakeUpState( )
```

**Retourne :**

soit Y\_WAKEUPSTATE\_SLEEPING, soit Y\_WAKEUPSTATE\_AWAKE

En cas d'erreur, déclenche une exception ou retourne Y\_WAKEUPSTATE\_INVALID.

## wakeupmonitor→isOnline()wakeupmonitor.isOnline()

YWakeUpMonitor

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du moniteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le moniteur est joignable, false sinon

**wakeupmonitor**→**load()****wakeupmonitor.load()****YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor**→**nextWakeUpMonitor()**  
**wakeupmonitor.nextWakeUpMonitor()**

**YWakeUpMonitor**

---

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

def **nextWakeUpMonitor()**

**Retourne :**

un pointeur sur un objet `YWakeUpMonitor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**wakeupmonitor**→**registerValueCallback()**  
**wakeupmonitor.registerValueCallback()**

---

**YWakeUpMonitor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**wakeupmonitor→resetSleepCountDown()**  
**wakeupmonitor.resetSleepCountDown()**

---

**YWakeUpMonitor**

Réinitialise le compteur de mise en sommeil.

```
def resetSleepCountDown( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



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

**YWakeUpMonitor**

Modifie le nom logique du moniteur.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du moniteur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set\_nextWakeUp()

YWakeUpMonitor

wakeupmonitor→setNextWakeUp()

wakeupmonitor.set\_nextWakeUp()

---

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
def set_nextWakeUp( newval)
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**wakeupmonitor**→**set\_powerDuration()****YWakeUpMonitor****wakeupmonitor**→**setPowerDuration()****wakeupmonitor.set\_powerDuration()**

---

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
def set_powerDuration( newval)
```

**Paramètres :**

**newval** un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→set\_sleepCountdown()**  
**wakeupmonitor→setSleepCountdown()**  
**wakeupmonitor.set\_sleepCountdown()**

---

**YWakeUpMonitor**

Modifie le temps avant le prochain sommeil .

```
def set_sleepCountdown( newval)
```

**Paramètres :**

**newval** un entier représentant le temps avant le prochain sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→set\_userdata()****YWakeUpMonitor****wakeupmonitor→setUserData()****wakeupmonitor.set\_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

---

**wakeupmonitor**→**sleep()****wakeupmonitor.sleep()**

**YWakeUpMonitor**

---

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
def sleep( secBeforeSleep)
```

**Paramètres :**

**secBeforeSleep** nombre de seconde avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## wakeupmonitor→sleepFor() wakeupmonitor.sleepFor()

## YWakeUpMonitor

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
def sleepFor( secUntilWakeUp, secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

### Paramètres :

**secUntilWakeUp** durée de la mise en sommeil, en secondes

**secBeforeSleep** nombre de secondes avant la mise en sommeil

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## wakeupmonitor→sleepUntil() wakeupmonitor.sleepUntil()

## YWakeUpMonitor

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
def sleepUntil( wakeUpTime, secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

### Paramètres :

**wakeUpTime** date/heure du réveil (format UNIX)

**secBeforeSleep** nombre de secondes avant la mise en sommeil

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**wakeupmonitor**→**wakeUp()****wakeupmonitor.wakeUp()****YWakeUpMonitor**

---

Force un réveil.

```
def wakeUp( )
```

### 3.44. Interface de la fonction WakeUpSchedule

La fonction WakeUpSchedule implémente une condition de réveil. Le réveil est spécifiée par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
<b>yFindWakeUpSchedule(func)</b> Permet de retrouver un réveil agendé d'après un identifiant donné.
<b>yFirstWakeUpSchedule()</b> Commence l'énumération des réveils agendés accessibles par la librairie.
Méthodes des objets YWakeUpSchedule
<b>wakeupschedule→describe()</b> Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format TYPE ( NAME ) = SERIAL . FUNCTIONID.
<b>wakeupschedule→get_advertisedValue()</b> Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).
<b>wakeupschedule→get_errorMessage()</b> Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.
<b>wakeupschedule→get_errorType()</b> Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.
<b>wakeupschedule→get_friendlyName()</b> Retourne un identifiant global du réveil agendé au format NOM_MODULE . NOM_FONCTION.
<b>wakeupschedule→get_functionDescriptor()</b> Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>wakeupschedule→get_functionId()</b> Retourne l'identifiant matériel du réveil agendé, sans référence au module.
<b>wakeupschedule→get_hardwareId()</b> Retourne l'identifiant matériel unique du réveil agendé au format SERIAL . FUNCTIONID.
<b>wakeupschedule→get_hours()</b> Retourne les heures où le réveil est actif..
<b>wakeupschedule→get_logicalName()</b> Retourne le nom logique du réveil agendé.
<b>wakeupschedule→get_minutes()</b> Retourne toutes les minutes de chaque heure où le réveil est actif.
<b>wakeupschedule→get_minutesA()</b>

Retourne les minutes de l'intervall 00-29 de chaque heures où le réveil est actif.

#### **wakeupschedule→get\_minutesB()**

Retourne les minutes de l'intervall 30-59 de chaque heure où le réveil est actif.

#### **wakeupschedule→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **wakeupschedule→get\_module\_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

#### **wakeupschedule→get\_monthDays()**

Retourne les jours du mois où le réveil est actif..

#### **wakeupschedule→get\_months()**

Retourne les mois où le réveil est actif..

#### **wakeupschedule→get\_nextOccurence()**

Retourne la date/heure de la prochaine occurrence de réveil

#### **wakeupschedule→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

#### **wakeupschedule→get\_weekDays()**

Retourne les jours de la semaine où le réveil est actif..

#### **wakeupschedule→isOnline()**

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

#### **wakeupschedule→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

#### **wakeupschedule→load(msValidity)**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

#### **wakeupschedule→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

#### **wakeupschedule→nextWakeUpSchedule()**

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

#### **wakeupschedule→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **wakeupschedule→set\_hours(newval, newval)**

Modifie les heures où un réveil doit avoir lieu

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

Modifie le nom logique du réveil agendé.

#### **wakeupschedule→set\_minutes(bitmap)**

Modifie toutes les minutes où un réveil doit avoir lieu

#### **wakeupschedule→set\_minutesA(newval, newval)**

Modifie les minutes de l'intervall 00-29 où un réveil doit avoir lieu

#### **wakeupschedule→set\_minutesB(newval)**

Modifie les minutes de l'intervall 30-59 où un réveil doit avoir lieu.

#### **wakeupschedule→set\_monthDays(newval, newval)**

Modifie les jours du mois où un réveil doit avoir lieu

#### **wakeupschedule→set\_months(newval, newval)**

Modifie les mois où un réveil doit avoir lieu

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

### 3. Reference

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

**wakeupschedule**→**set\_weekDays**(**newval**, **newval**)

Modifie les jours de la semaine où un réveil doit avoir lieu

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YWakeUpSchedule.FindWakeUpSchedule() yFindWakeUpSchedule() YWakeUpSchedule.FindWakeUpSchedule()

## YWakeUpSchedule

Permet de retrouver un réveil agendé d'après un identifiant donné.

```
def FindWakeUpSchedule( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le réveil agendé soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpSchedule.IsOnline()` pour tester si le réveil agendé est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le réveil agendé sans ambiguïté

### Retourne :

un objet de classe `YWakeUpSchedule` qui permet ensuite de contrôler le réveil agendé.

**YWakeUpSchedule.FirstWakeUpSchedule()**  
**yFirstWakeUpSchedule()**  
**YWakeUpSchedule.FirstWakeUpSchedule()**

---

**YWakeUpSchedule**

Commence l'énumération des réveils agendés accessibles par la librairie.

```
def FirstWakeUpSchedule( )
```

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule( )` pour itérer sur les autres réveils agendés.

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule`, correspondant au premier réveil agendé accessible en ligne, ou `null` si il n'y a pas de réveils agendés disponibles.

---

**wakeupschedule→describe()**  
**wakeupschedule.describe()**

---

**YWakeUpSchedule**

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant le réveil agendé (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**wakeupschedule→get\_advertisedValue()**

**YWakeUpSchedule**

**wakeupschedule→advertisedValue()**

**wakeupschedule.get\_advertisedValue()**

---

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

def **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante du réveil agendé (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.



---

**wakeupschedule→get\_errorMessage()****YWakeUpSchedule****wakeupschedule→errorMessage()****wakeupschedule.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

**wakeupschedule→get\_errorType()**

**YWakeUpSchedule**

**wakeupschedule→errorType()**

**wakeupschedule.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

---

**wakeupschedule→get\_friendlyName()****YWakeUpSchedule****wakeupschedule→friendlyName()****wakeupschedule.get\_friendlyName()**

---

Retourne un identifiant global du réveil agendé au format `NOM_MODULE . NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du réveil agendé si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du réveil agendé (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le réveil agendé en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**wakeupschedule→get\_functionDescriptor()**

**YWakeUpSchedule**

**wakeupschedule→functionDescriptor()**

**wakeupschedule.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**wakeupschedule→get\_functionId()****YWakeUpSchedule****wakeupschedule→functionId()****wakeupschedule.get\_functionId()**

---

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le réveil agendé (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**wakeupschedule**→**get\_hardwareId()**

**YWakeUpSchedule**

**wakeupschedule**→**hardwareId()**

**wakeupschedule.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du réveil agendé au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du réveil agendé (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le réveil agendé (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**wakeupschedule→get\_hours()****YWakeUpSchedule****wakeupschedule→hours()****wakeupschedule.get\_hours()**

---

Retourne les heures où le réveil est actif..

```
def get_hours( )
```

**Retourne :**

un entier représentant les heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_HOURS\_INVALID.

**wakeupschedule→get\_logicalName()**

**YWakeUpSchedule**

**wakeupschedule→logicalName()**

**wakeupschedule.get\_logicalName()**

---

Retourne le nom logique du réveil agendé.

def **get\_logicalName**( )

**Retourne :**

une chaîne de caractères représentant le nom logique du réveil agendé. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.



---

**wakeupschedule→get\_minutes()****YWakeUpSchedule****wakeupschedule→minutes()****wakeupschedule.get\_minutes()**

---

Retourne toutes les minutes de chaque heure où le réveil est actif.

```
def get_minutes( )
```

**wakeupschedule→get\_minutesA()**

**YWakeUpSchedule**

**wakeupschedule→minutesA()**

**wakeupschedule.get\_minutesA()**

---

Retourne les minutes de l'interval 00-29 de chaque heures où le réveil est actif.

```
def get_minutesA( )
```

**Retourne :**

un entier représentant les minutes de l'interval 00-29 de chaque heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MINUTESA\_INVALID.

---

**wakeupschedule→get\_minutesB()****YWakeUpSchedule****wakeupschedule→minutesB()****wakeupschedule.get\_minutesB()**

---

Retourne les minutes de l'intervall 30-59 de chaque heure où le réveil est actif.

```
def get_minutesB( )
```

**Retourne :**

un entier représentant les minutes de l'intervall 30-59 de chaque heure où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MINUTESB\_INVALID.

**wakeupschedule**→**get\_module()**

**YWakeUpSchedule**

**wakeupschedule**→**module()**

**wakeupschedule.get\_module()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

---

**wakeupschedule→get\_monthDays()****YWakeUpSchedule****wakeupschedule→monthDays()****wakeupschedule.get\_monthDays()**

---

Retourne les jours du mois où le réveil est actif..

```
def get_monthDays( )
```

**Retourne :**

un entier représentant les jours du mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MONTHDAYS\_INVALID.

**wakeupschedule→get\_months()**

**YWakeUpSchedule**

**wakeupschedule→months()**

**wakeupschedule.get\_months()**

---

Retourne les mois où le réveil est actif..

```
def get_months( )
```

**Retourne :**

un entier représentant les mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MONTHS\_INVALID.

---

**wakeupschedule→get\_nextOccurence()****YWakeUpSchedule****wakeupschedule→nextOccurence()****wakeupschedule.get\_nextOccurence()**

---

Retourne la date/heure de la prochaine occurrence de réveil

```
def get_nextOccurence( )
```

**Retourne :**

un entier représentant la date/heure de la prochaine occurrence de réveil

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTOCCURENCE\_INVALID.

**wakeupschedule→get\_userData()**

**YWakeUpSchedule**

**wakeupschedule→userData()**

**wakeupschedule.get\_userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

```
def get_userData( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.



---

**wakeupschedule→get\_weekDays()****YWakeUpSchedule****wakeupschedule→weekDays()****wakeupschedule.get\_weekDays()**

---

Retourne les jours de la semaine où le réveil est actif..

```
def get_weekDays( )
```

**Retourne :**

un entier représentant les jours de la semaine où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_WEEKDAYS\_INVALID.

**wakeupschedule**→**isOnline()**  
**wakeupschedule.isOnline()**

---

**YWakeUpSchedule**

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du réveil agendé sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le réveil agendé est joignable, `false` sinon

**wakeupschedule→load()wakeupschedule.load()****YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule**→**nextWakeUpSchedule()**  
**wakeupschedule.nextWakeUpSchedule()**

**YWakeUpSchedule**

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

def **nextWakeUpSchedule()**

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**wakeupschedule→registerValueCallback()**  
**wakeupschedule.registerValueCallback()**

---

**YWakeUpSchedule**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**wakeupschedule→set\_hours()**

**YWakeUpSchedule**

**wakeupschedule→setHours()**

**wakeupschedule.set\_hours()**

---

Modifie les heures où un réveil doit avoir lieu

```
def set_hours( newval)
```

**Paramètres :**

**newval** un entier représentant les heures où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**wakeupschedule→set\_logicalName()****YWakeUpSchedule****wakeupschedule→setLogicalName()****wakeupschedule.set\_logicalName()**

---

Modifie le nom logique du réveil agendé.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du réveil agendé.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_minutes()**

**YWakeUpSchedule**

**wakeupschedule→setMinutes()**

**wakeupschedule.set\_minutes()**

---

Modifie toutes les minutes où un réveil doit avoir lieu

```
def set_minutes( bitmap)
```

**Paramètres :**

**bitmap** Minutes 00-59 de chaque heure où le réveil est actif.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



---

**wakeupschedule→set\_minutesA()****YWakeUpSchedule****wakeupschedule→setMinutesA()****wakeupschedule.set\_minutesA()**

---

Modifie les minutes de l'intervall 00-29 où un réveil doit avoir lieu

```
def set_minutesA( newval)
```

**Paramètres :**

**newval** un entier représentant les minutes de l'intervall 00-29 où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_minutesB()**

**YWakeUpSchedule**

**wakeupschedule→setMinutesB()**

**wakeupschedule.set\_minutesB()**

---

Modifie les minutes de l'interval 30-59 où un réveil doit avoir lieu.

```
def set_minutesB( newval)
```

**Paramètres :**

**newval** un entier représentant les minutes de l'interval 30-59 où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**wakeupschedule→set\_monthDays()****YWakeUpSchedule****wakeupschedule→setMonthDays()****wakeupschedule.set\_monthDays()**

---

Modifie les jours du mois où un réveil doit avoir lieu

```
def set_monthDays( newval)
```

**Paramètres :**

**newval** un entier représentant les jours du mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_months()**

**YWakeUpSchedule**

**wakeupschedule→setMonths()**

**wakeupschedule.set\_months()**

---

Modifie les mois où un réveil doit avoir lieu

```
def set_months( newval)
```

**Paramètres :**

**newval** un entier représentant les mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**wakeupschedule→set\_userdata()****YWakeUpSchedule****wakeupschedule→setUserData()****wakeupschedule.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**wakeupschedule→set\_weekDays()**

**YWakeUpSchedule**

**wakeupschedule→setWeekDays()**

**wakeupschedule.set\_weekDays()**

---

Modifie les jours de la semaine où un réveil doit avoir lieu

```
def set_weekDays( newval)
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.45. Interface de la fonction Watchdog

La fonction WatchDog est gérée comme un relais qui couperait brièvement l'alimentation d'un appareil après un d'attente temps donné afin de provoquer une réinitialisation complète de cet appareil. Il suffit d'appeler le watchdog à intervalle régulier pour l'empêcher de provoquer la réinitialisation. Le watchdog peut aussi être piloté directement à l'aide des méthode *pulse* et *delayedpulse* pour éteindre un appareil pendant un temps donné.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindWatchdog(func)

Permet de retrouver un watchdog d'après un identifiant donné.

#### yFirstWatchdog()

Commence l'énumération des watchdog accessibles par la librairie.

### Méthodes des objets YWatchdog

#### watchdog→delayedPulse(ms\_delay, ms\_duration)

Pré-programme une impulsion

#### watchdog→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

#### watchdog→get\_advertisedValue()

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

#### watchdog→get\_autoStart()

Retourne l'état du watchdog à la mise sous tension du module.

#### watchdog→get\_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

#### watchdog→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

#### watchdog→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

#### watchdog→get\_friendlyName()

Retourne un identifiant global du watchdog au format `NOM_MODULE . NOM_FONCTION`.

#### watchdog→get\_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

#### watchdog→get\_functionId()

Retourne l'identifiant matériel du watchdog, sans référence au module.

#### **watchdog→get\_hardwareId()**

Retourne l'identifiant matériel unique du watchdog au format SERIAL.FUNCTIONID.

#### **watchdog→get\_logicalName()**

Retourne le nom logique du watchdog.

#### **watchdog→get\_maxTimeOnStateA()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

#### **watchdog→get\_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

#### **watchdog→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **watchdog→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **watchdog→get\_output()**

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

#### **watchdog→get\_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

#### **watchdog→get\_running()**

Retourne l'état du watchdog.

#### **watchdog→get\_state()**

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

#### **watchdog→get\_stateAtPowerOn()**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

#### **watchdog→get\_triggerDelay()**

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

#### **watchdog→get\_triggerDuration()**

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

#### **watchdog→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

#### **watchdog→isOnline()**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

#### **watchdog→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

#### **watchdog→load(msValidity)**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

#### **watchdog→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

#### **watchdog→nextWatchdog()**

Continue l'énumération des watchdog commencée à l'aide de yFirstWatchdog().

#### **watchdog→pulse(ms\_duration)**

Commute le relais à l'état B (actif) pour une durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).



**watchdog→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**watchdog→resetWatchdog()**

Réinitialise le WatchDog.

**watchdog→set\_autoStart(newval)**

Modifie l'état du watching au démarrage du module.

**watchdog→set\_logicalName(newval)**

Modifie le nom logique du watchdog.

**watchdog→set\_maxTimeOnStateA(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**watchdog→set\_maxTimeOnStateB(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

**watchdog→set\_output(newval)**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

**watchdog→set\_running(newval)**

Modifie manuellement l'état de fonctionnement du watchdog.

**watchdog→set\_state(newval)**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

**watchdog→set\_stateAtPowerOn(newval)**

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

**watchdog→set\_triggerDelay(newval)**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

**watchdog→set\_triggerDuration(newval)**

Modifie la durée des resets générés par le watchdog, en millisecondes.

**watchdog→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userdata.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YWatchdog.FindWatchdog()****YWatchdog****yFindWatchdog()YWatchdog.FindWatchdog()**

Permet de retrouver un watchdog d'après un identifiant donné.

```
def FindWatchdog( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le watchdog soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWatchdog.isOnline()` pour tester si le watchdog est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le watchdog sans ambiguïté

**Retourne :**

un objet de classe `YWatchdog` qui permet ensuite de contrôler le watchdog.

**YWatchdog.FirstWatchdog()****YWatchdog****yFirstWatchdog()YWatchdog.FirstWatchdog()**

Commence l'énumération des watchdog accessibles par la librairie.

```
def FirstWatchdog( )
```

Utiliser la fonction `YWatchdog.nextWatchdog( )` pour itérer sur les autres watchdog.

**Retourne :**

un pointeur sur un objet `YWatchdog`, correspondant au premier watchdog accessible en ligne, ou `null` si il n'y a pas de watchdog disponibles.

**watchdog→delayedPulse()****watchdog.delayedPulse()**

**YWatchdog**

---

Pré-programme une impulsion

```
def delayedPulse( ms_delay, ms_duration)
```

**Paramètres :**

**ms\_delay**     délai d'attente avant l'impulsion, en millisecondes

**ms\_duration**   durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→describe()watchdog.describe()****YWatchdog**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

**Retourne :**

une chaîne de caractères décrivant le watchdog (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**watchdog**→**get\_advertisedValue()**

**YWatchdog**

**watchdog**→**advertisedValue()**

**watchdog.get\_advertisedValue()**

---

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du watchdog (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**watchdog→get\_autoStart()****YWatchdog****watchdog→autoStart()watchdog.get\_autoStart()**

Retourne l'état du watchdog à la mise sous tension du module.

```
def get_autoStart( )
```

**Retourne :**

soit Y\_AUTOSTART\_OFF, soit Y\_AUTOSTART\_ON, selon l'état du watchdog à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_AUTOSTART\_INVALID.

**watchdog→get\_countdown()**

**YWatchdog**

**watchdog→countdown()watchdog.get\_countdown()**

---

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

```
def get_countdown( )
```

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.



---

**watchdog→get\_errorMessage()****YWatchdog****watchdog→errorMessage()****watchdog.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

**watchdog**→**get\_errorType()**

**YWatchdog**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

---

**watchdog→get\_friendlyName()****YWatchdog****watchdog→friendlyName()****watchdog.get\_friendlyName()**

---

Retourne un identifiant global du watchdog au format `NOM_MODULE . NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du watchdog si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du watchdog (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant le watchdog en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

**watchdog**→**get\_functionDescriptor()**

**YWatchdog**

**watchdog**→**functionDescriptor()**

**watchdog.get\_functionDescriptor()**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**watchdog→get\_functionId()****YWatchdog****watchdog→functionId()watchdog.get\_functionId()**

---

Retourne l'identifiant matériel du watchdog, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**watchdog**→**get\_hardwareId()**

**YWatchdog**

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

---

Retourne l'identifiant matériel unique du watchdog au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du watchdog (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**watchdog→get\_logicalName()****YWatchdog****watchdog→logicalName()****watchdog.get\_logicalName()**

---

Retourne le nom logique du watchdog.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du watchdog. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**watchdog**→**get\_maxTimeOnStateA()**

**YWatchdog**

**watchdog**→**maxTimeOnStateA()**

**watchdog.get\_maxTimeOnStateA()**

---

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
def get_maxTimeOnStateA( )
```

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEA\_INVALID.



---

**watchdog→get\_maxTimeOnStateB()****YWatchdog****watchdog→maxTimeOnStateB()****watchdog.get\_maxTimeOnStateB()**

---

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
def get_maxTimeOnStateB( )
```

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEB\_INVALID.

**watchdog**→**get\_module()**

**YWatchdog**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

**watchdog→get\_output()****YWatchdog****watchdog→output()watchdog.get\_output()**

---

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
def get_output( )
```

**Retourne :**

soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUT\_INVALID.

**watchdog→get\_pulseTimer()**

**YWatchdog**

**watchdog→pulseTimer()watchdog.get\_pulseTimer()**

---

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

def **get\_pulseTimer()**

Si aucune impulsion n'est en cours, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne Y\_PULSETIMER\_INVALID.

---

**watchdog→get\_running()****YWatchdog****watchdog→running()watchdog.get\_running()**

---

Retourne l'état du watchdog.

```
def get_running( )
```

**Retourne :**

soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon l'état du watchdog

En cas d'erreur, déclenche une exception ou retourne Y\_RUNNING\_INVALID.

**watchdog→get\_state()**

**YWatchdog**

**watchdog→state()watchdog.get\_state()**

---

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
def get_state( )
```

**Retourne :**

soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y\_STATE\_INVALID.

---

**watchdog→get\_stateAtPowerOn()****YWatchdog****watchdog→stateAtPowerOn()****watchdog.get\_stateAtPowerOn()**

---

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
def get_stateAtPowerOn( )
```

**Retourne :**

une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B représentant l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y\_STATEATPOWERON\_INVALID.

**watchdog→get\_triggerDelay()**

**YWatchdog**

**watchdog→triggerDelay()**

**watchdog.get\_triggerDelay()**

---

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

```
def get_triggerDelay( )
```

**Retourne :**

un entier représentant le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_TRIGGERDELAY\_INVALID.



---

**watchdog→get\_triggerDuration()****YWatchdog****watchdog→triggerDuration()****watchdog.get\_triggerDuration()**

---

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

```
def get_triggerDuration( )
```

**Retourne :**

un entier représentant la durée d'un reset généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_TRIGGERDURATION\_INVALID.

**watchdog**→**get\_userdata()**

**YWatchdog**

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

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

---

**watchdog→isOnline()watchdog.isOnline()****YWatchdog**

---

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache du watchdog sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le watchdog est joignable, `false` sinon

**watchdog→load()watchdog.load()****YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**watchdog**→**nextWatchdog()**  
**watchdog.nextWatchdog()**

---

**YWatchdog**

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

```
def nextWatchdog( )
```

**Retourne :**

un pointeur sur un objet `YWatchdog` accessible en ligne, ou `null` lorsque l'énumération est terminée.

## watchdog→pulse()watchdog.pulse()

YWatchdog

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
def pulse( ms_duration)
```

### Paramètres :

**ms\_duration** durée de l'impulsion, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**watchdog→registerValueCallback()**  
**watchdog.registerValueCallback()**

---

**YWatchdog**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**watchdog→resetWatchdog()**  
**watchdog.resetWatchdog()**

---

**YWatchdog**

Réinitialise le WatchDog.

```
def resetWatchdog( )
```

Quand le watchdog est en fonctionnement cette fonction doit être appelée à interval régulier, pour empêcher que le watdog ne se déclenche

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**watchdog→set\_autoStart()****YWatchdog****watchdog→setAutoStart()watchdog.set\_autoStart()**

Modifie l'état du watching au démarrage du module.

```
def set_autoStart( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watching au démarrage du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_logicalName()**

**YWatchdog**

**watchdog→setLogicalName()**

**watchdog.set\_logicalName()**

---

Modifie le nom logique du watchdog.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du watchdog.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**watchdog→set\_maxTimeOnStateA()****YWatchdog****watchdog→setMaxTimeOnStateA()****watchdog.set\_maxTimeOnStateA()**

---

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
def set_maxTimeOnStateA( newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_maxTimeOnStateB()**

**YWatchdog**

**watchdog→setMaxTimeOnStateB()**

**watchdog.set\_maxTimeOnStateB()**

---

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
def set_maxTimeOnStateB( newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_output()****YWatchdog****watchdog→setOutput()watchdog.set\_output()**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
def set_output( newval)
```

**Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog**→**set\_running()**

**YWatchdog**

**watchdog**→**setRunning()****watchdog.set\_running()**

---

Modifie manuellement l'état de fonctionnement du watchdog.

```
def set_running( newval)
```

**Paramètres :**

**newval** soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon manuellement l'état de fonctionnement du watchdog

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_state()****YWatchdog****watchdog→setState()watchdog.set\_state()**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
def set_state( newval)
```

**Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_stateAtPowerOn()**

**YWatchdog**

**watchdog→setStateAtPowerOn()**

**watchdog.set\_stateAtPowerOn()**

---

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
def set_stateAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**watchdog→set\_triggerDelay()**  
**watchdog→setTriggerDelay()**  
**watchdog.set\_triggerDelay()**

**YWatchdog**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

```
def set_triggerDelay( newval)
```

**Paramètres :**

**newval** un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDuration()**  
**watchdog→setTriggerDuration()**  
**watchdog.set\_triggerDuration()**

---

**YWatchdog**

Modifie la durée des resets générés par le watchdog, en millisecondes.

```
def set_triggerDuration( newval)
```

**Paramètres :**

**newval** un entier représentant la durée des resets générés par le watchdog, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**watchdog→set\_userdata()****YWatchdog****watchdog→setUserData()watchdog.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

### 3.46. Interface de la fonction Wireless

La fonction YWireless permet de configurer et de contrôler la configuration du réseau sans fil sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
<b>yFindWireless(func)</b> Permet de retrouver une interface réseau sans fil d'après un identifiant donné.
<b>yFirstWireless()</b> Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.
Méthodes des objets YWireless
<b>wireless→adhocNetwork(ssid, securityKey)</b> Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".
<b>wireless→describe()</b> Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE ( NAME ) = SERIAL . FUNCTIONID.
<b>wireless→get_advertisedValue()</b> Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).
<b>wireless→get_channel()</b> Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.
<b>wireless→get_detectedWlans()</b> Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.
<b>wireless→get_errorMessage()</b> Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.
<b>wireless→get_errorType()</b> Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.
<b>wireless→get_friendlyName()</b> Retourne un identifiant global de l'interface réseau sans fil au format NOM_MODULE . NOM_FONCTION.
<b>wireless→get_functionDescriptor()</b> Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>wireless→get_functionId()</b> Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.
<b>wireless→get_hardwareId()</b>

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format `SERIAL.FUNCTIONID`.

**wireless→get\_linkQuality()**

Retourne la qualité de la connection, exprimée en pourcents.

**wireless→get\_logicalName()**

Retourne le nom logique de l'interface réseau sans fil.

**wireless→get\_message()**

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

**wireless→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

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

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**wireless→get\_security()**

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

**wireless→get\_ssid()**

Retourne le nom (SSID) du réseau sans-fil sélectionné.

**wireless→get\_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

**wireless→isOnline()**

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

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

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

**wireless→load(msValidity)**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

**wireless→nextWireless()**

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

**wireless→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**wireless→set\_logicalName(newval)**

Modifie le nom logique de l'interface réseau sans fil.

**wireless→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YWireless.FindWireless() yFindWireless()YWireless.FindWireless()

YWireless

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

```
def FindWireless( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau sans fil soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWireless.isOnline()` pour tester si l'interface réseau sans fil est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'interface réseau sans fil sans ambiguïté

### Retourne :

un objet de classe `YWireless` qui permet ensuite de contrôler l'interface réseau sans fil.

## **YWireless.FirstWireless() yFirstWireless()YWireless.FirstWireless()**

## **YWireless**

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

```
def FirstWireless( )
```

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

**Retourne :**

un pointeur sur un objet `YWireless`, correspondant à la première interface réseau sans fil accessible en ligne, ou `null` si il n'y a pas de interfaces réseau sans fil disponibles.

**wireless→adhocNetwork()****wireless.adhocNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

```
def adhocNetwork( ssid, securityKey)
```

Si une clef d'accès est spécifiée, le réseau sera protégé par une sécurité WEP128 (l'utilisation de WPA n'est pas standardisée en mode ad-hoc). N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à créer  
**securityKey** clé d'accès de réseau, sous forme de chaîne de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.



**wireless→describe()wireless.describe()****YWireless**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
def describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

**Retourne :**

une chaîne de caractères décrivant l'interface réseau sans fil (ex:  
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

**wireless→get\_advertisedValue()**

**YWireless**

**wireless→advertisedValue()**

**wireless.get\_advertisedValue()**

---

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

```
def get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wireless**→**get\_channel()****YWireless****wireless**→**channel()****wireless.get\_channel()**

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

```
def get_channel( )
```

**Retourne :**

un entier représentant le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé

En cas d'erreur, déclenche une exception ou retourne `Y_CHANNEL_INVALID`.

**wireless→get\_detectedWlans()**

**YWireless**

**wireless→detectedWlans()**

**wireless.get\_detectedWlans()**

---

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

```
def get_detectedWlans( )
```

La liste n'est pas mise à jour quand le module est déjà connecté à un accès sans fil (mode "infrastructure"). Pour forcer la détection des réseaux sans fil, il faut appeler `adhocNetwork( )` pour se déconnecter du réseau actuel. L'appelant est responsable de la désallocation de la liste retournée.

**Retourne :**

une liste d'objets `YWlanRecord`, contenant le SSID, le canal, la qualité du signal, et l'algorithme de sécurité utilisé par le réseau sans-fil

En cas d'erreur, déclenche une exception ou retourne une liste vide.

---

**wireless→get\_errorMessage()****YWireless****wireless→errorMessage()****wireless.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

```
def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

**wireless**→**get\_errorType()**

**YWireless**

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

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

```
def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

**wireless**→**get\_friendlyName()****YWireless****wireless**→**friendlyName()****wireless.get\_friendlyName()**

Retourne un identifiant global de l'interface réseau sans fil au format `NOM_MODULE.NOM_FONCTION`.

```
def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'interface réseau sans fil si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau sans fil (par exemple: `MyCustomName.relay1`)

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

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

---

**YWireless**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID



**wireless**→**get\_functionId()****YWireless****wireless**→**functionId()****wireless.get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

```
def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**wireless**→**get\_hardwareId()**

**YWireless**

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

---

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format `SERIAL.FUNCTIONID`.

```
def get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau sans fil (par exemple `RELAYLO1-123456.relay1`).

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

**wireless**→**get\_linkQuality()****YWireless****wireless**→**linkQuality()****wireless.get\_linkQuality()**

Retourne la qualité de la connexion, exprimée en pourcents.

```
def get_linkQuality( )
```

**Retourne :**

un entier représentant la qualité de la connexion, exprimée en pourcents

En cas d'erreur, déclenche une exception ou retourne `Y_LINKQUALITY_INVALID`.

**wireless**→**get\_logicalName()**

**YWireless**

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

---

Retourne le nom logique de l'interface réseau sans fil.

```
def get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau sans fil. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**wireless**→**get\_message()****YWireless****wireless**→**message()****wireless.get\_message()**

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

```
def get_message( )
```

**Retourne :**

une chaîne de caractères représentant le dernier message de diagnostic de l'interface au réseau sans fil

En cas d'erreur, déclenche une exception ou retourne Y\_MESSAGE\_INVALID.

**wireless**→**get\_module()**

**YWireless**

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

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
def get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**wireless**→**get\_security()****YWireless****wireless**→**security()****wireless.get\_security()**

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

```
def get_security( )
```

**Retourne :**

une valeur parmi Y\_SECURITY\_UNKNOWN, Y\_SECURITY\_OPEN, Y\_SECURITY\_WEP, Y\_SECURITY\_WPA et Y\_SECURITY\_WPA2 représentant l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y\_SECURITY\_INVALID.

**wireless**→**get\_ssid()**

**YWireless**

**wireless**→**ssid()****wireless.get\_ssid()**

---

Retourne le nom (SSID) du réseau sans-fil sélectionné.

```
def get_ssid( )
```

**Retourne :**

une chaîne de caractères représentant le nom (SSID) du réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y\_SSID\_INVALID.



---

**wireless**→**get\_userdata()****YWireless****wireless**→**userData()****wireless.get\_userdata()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

```
def get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## wireless→isOnline()**wireless.isOnline()**

**YWireless**

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

```
def isOnline( )
```

Si les valeurs des attributs en cache de l'interface réseau sans fil sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'interface réseau sans fil est joignable, `false` sinon

**wireless→joinNetwork()****wireless.joinNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

```
def joinNetwork( ssid, securityKey)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à utiliser

**securityKey** clé d'accès au réseau, sous forme de chaîne de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→load()wireless.load()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**wireless**→**nextWireless()****wireless.nextWireless()****YWireless**

---

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

```
def nextWireless( )
```

**Retourne :**

un pointeur sur un objet `YWireless` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wireless→registerValueCallback()**  
**wireless.registerValueCallback()****YWireless**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

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

**YWireless**

Modifie le nom logique de l'interface réseau sans fil.

```
def set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless**→**set\_userdata()**

**YWireless**

**wireless**→**setUserData()****wireless.set\_userdata()**

---

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
def set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser



# Index

## A

Accelerometer 32  
adhocNetwork, YWireless 1563  
Alimentation 450  
AnButton 74

## B

Blueprint 10  
Brute 312

## C

calibrate, YLightSensor 695  
calibrateFromPoints, YAccelerometer 36  
calibrateFromPoints, YCarbonDioxide 116  
calibrateFromPoints, YCompass 184  
calibrateFromPoints, YCurrent 224  
calibrateFromPoints, YGenericSensor 507  
calibrateFromPoints, YGyro 553  
calibrateFromPoints, YHumidity 629  
calibrateFromPoints, YLightSensor 696  
calibrateFromPoints, YMagnetometer 735  
calibrateFromPoints, YPower 906  
calibrateFromPoints, YPressure 949  
calibrateFromPoints, YQt 1049  
calibrateFromPoints, YSensor 1187  
calibrateFromPoints, YTemperature 1261  
calibrateFromPoints, YTilt 1302  
calibrateFromPoints, YVoc 1341  
calibrateFromPoints, YVotage 1380  
callbackLogin, YNetwork 827  
cancel3DCalibration, YRefFrame 1115  
CarbonDioxide 112  
CheckLogicalName, YAPI 12  
clear, YDisplayLayer 419  
clearConsole, YDisplayLayer 420  
ColorLed 151  
Compass 180  
Configuration 1111  
consoleOut, YDisplayLayer 421  
Contrôle 4, 5, 450, 779, 879  
copyLayerContent, YDisplay 375  
Current 220

## D

DataLogger 259  
delayedPulse, YDigitalIO 331  
delayedPulse, YRelay 1151  
delayedPulse, YWatchdog 1519  
describe, YAccelerometer 37  
describe, YAnButton 78  
describe, YCarbonDioxide 117  
describe, YColorLed 154

describe, YCompass 185  
describe, YCurrent 225  
describe, YDataLogger 262  
describe, YDigitalIO 332  
describe, YDisplay 376  
describe, YDualPower 453  
describe, YFiles 478  
describe, YGenericSensor 508  
describe, YGyro 554  
describe, YHubPort 603  
describe, YHumidity 630  
describe, YLed 667  
describe, YLightSensor 697  
describe, YMagnetometer 736  
describe, YModule 783  
describe, YNetwork 828  
describe, YOsControl 882  
describe, YPower 907  
describe, YPressure 950  
describe, YPwmOutput 988  
describe, YPwmPowerSource 1025  
describe, YQt 1050  
describe, YRealTimeClock 1087  
describe, YRefFrame 1116  
describe, YRelay 1152  
describe, YSensor 1188  
describe, YServo 1226  
describe, YTemperature 1262  
describe, YTilt 1303  
describe, YVoc 1342  
describe, YVotage 1381  
describe, YWakeUpMonitor 1447  
describe, YWakeUpSchedule 1482  
describe, YWatchdog 1520  
describe, YWireless 1564  
DigitalIO 327  
DisableExceptions, YAPI 13  
Display 371  
DisplayLayer 418  
Données 290, 300, 312  
download, YFiles 479  
download, YModule 784  
drawBar, YDisplayLayer 422  
drawBitmap, YDisplayLayer 423  
drawCircle, YDisplayLayer 424  
drawDisc, YDisplayLayer 425  
drawImage, YDisplayLayer 426  
drawPixel, YDisplayLayer 427  
drawRect, YDisplayLayer 428  
drawText, YDisplayLayer 429  
dutyCycleMove, YPwmOutput 989  
Dynamique 3

## E

EnableExceptions, YAPI 14  
Enregistrées 300, 312  
Erreurs 7

## F

fade, YDisplay 377  
Fichiers 3  
Files 475  
FindAccelerometer, YAccelerometer 34  
FindAnButton, YAnButton 76  
FindCarbonDioxide, YCarbonDioxide 114  
FindColorLed, YColorLed 152  
FindCompass, YCompass 182  
FindCurrent, YCurrent 222  
FindDataLogger, YDataLogger 260  
FindDigitalIO, YDigitalIO 329  
FindDisplay, YDisplay 373  
FindDualPower, YDualPower 451  
FindFiles, YFiles 476  
FindGenericSensor, YGenericSensor 505  
FindGyro, YGyro 551  
FindHubPort, YHubPort 601  
FindHumidity, YHumidity 627  
FindLed, YLed 665  
FindLightSensor, YLightSensor 693  
FindMagnetometer, YMagnetometer 733  
FindModule, YModule 781  
FindNetwork, YNetwork 825  
FindOsControl, YOsControl 880  
FindPower, YPower 904  
FindPressure, YPressure 947  
FindPwmOutput, YPwmOutput 986  
FindPwmPowerSource, YPwmPowerSource 1023  
FindQt, YQt 1047  
FindRealTimeClock, YRealTimeClock 1085  
FindRefFrame, YRefFrame 1113  
FindRelay, YRelay 1149  
FindSensor, YSensor 1185  
FindServo, YServo 1224  
FindTemperature, YTemperature 1259  
FindTilt, YTilt 1300  
FindVoc, YVoc 1339  
FindVoltage, YVoltage 1378  
FindVSource, YVSource 1416  
FindWakeUpMonitor, YWakeUpMonitor 1445  
FindWakeUpSchedule, YWakeUpSchedule 1480  
FindWatchdog, YWatchdog 1517  
FindWireless, YWireless 1561  
FirstAccelerometer, YAccelerometer 35  
FirstAnButton, YAnButton 77  
FirstCarbonDioxide, YCarbonDioxide 115  
FirstColorLed, YColorLed 153  
FirstCompass, YCompass 183  
FirstCurrent, YCurrent 223  
FirstDataLogger, YDataLogger 261  
FirstDigitalIO, YDigitalIO 330

FirstDisplay, YDisplay 374  
FirstDualPower, YDualPower 452  
FirstFiles, YFiles 477  
FirstGenericSensor, YGenericSensor 506  
FirstGyro, YGyro 552  
FirstHubPort, YHubPort 602  
FirstHumidity, YHumidity 628  
FirstLed, YLed 666  
FirstLightSensor, YLightSensor 694  
FirstMagnetometer, YMagnetometer 734  
FirstModule, YModule 782  
FirstNetwork, YNetwork 826  
FirstOsControl, YOsControl 881  
FirstPower, YPower 905  
FirstPressure, YPressure 948  
FirstPwmOutput, YPwmOutput 987  
FirstPwmPowerSource, YPwmPowerSource 1024  
FirstQt, YQt 1048  
FirstRealTimeClock, YRealTimeClock 1086  
FirstRefFrame, YRefFrame 1114  
FirstRelay, YRelay 1150  
FirstSensor, YSensor 1186  
FirstServo, YServo 1225  
FirstTemperature, YTemperature 1260  
FirstTilt, YTilt 1301  
FirstVoc, YVoc 1340  
FirstVoltage, YVoltage 1379  
FirstVSource, YVSource 1417  
FirstWakeUpMonitor, YWakeUpMonitor 1446  
FirstWakeUpSchedule, YWakeUpSchedule 1481  
FirstWatchdog, YWatchdog 1518  
FirstWireless, YWireless 1562  
Fonctions 11, 1183  
forgetAllDataStreams, YDataLogger 263  
format\_fs, YFiles 480  
Forme 290  
FreeAPI, YAPI 15  
functionCount, YModule 785  
functionId, YModule 786  
functionName, YModule 787  
functionValue, YModule 788

## G

GenericSensor 503  
get\_3DCalibrationHint, YRefFrame 1117  
get\_3DCalibrationLogMsg, YRefFrame 1118  
get\_3DCalibrationProgress, YRefFrame 1119  
get\_3DCalibrationStage, YRefFrame 1120  
get\_3DCalibrationStageProgress, YRefFrame 1121  
get\_adminPassword, YNetwork 829  
get\_advertisedValue, YAccelerometer 38  
get\_advertisedValue, YAnButton 79  
get\_advertisedValue, YCarbonDioxide 118  
get\_advertisedValue, YColorLed 155  
get\_advertisedValue, YCompass 186  
get\_advertisedValue, YCurrent 226  
get\_advertisedValue, YDataLogger 264

get\_advertisedValue, YDigitalIO 333  
 get\_advertisedValue, YDisplay 378  
 get\_advertisedValue, YDualPower 454  
 get\_advertisedValue, YFiles 481  
 get\_advertisedValue, YGenericSensor 509  
 get\_advertisedValue, YGyro 555  
 get\_advertisedValue, YHubPort 604  
 get\_advertisedValue, YHumidity 631  
 get\_advertisedValue, YLed 668  
 get\_advertisedValue, YLightSensor 698  
 get\_advertisedValue, YMagnetometer 737  
 get\_advertisedValue, YNetwork 830  
 get\_advertisedValue, YOsControl 883  
 get\_advertisedValue, YPower 908  
 get\_advertisedValue, YPressure 951  
 get\_advertisedValue, YPwmOutput 990  
 get\_advertisedValue, YPwmPowerSource 1026  
 get\_advertisedValue, YQt 1051  
 get\_advertisedValue, YRealTimeClock 1088  
 get\_advertisedValue, YRefFrame 1122  
 get\_advertisedValue, YRelay 1153  
 get\_advertisedValue, YSensor 1189  
 get\_advertisedValue, YServo 1227  
 get\_advertisedValue, YTemperature 1263  
 get\_advertisedValue, YTilt 1304  
 get\_advertisedValue, YVoc 1343  
 get\_advertisedValue, YVoltage 1382  
 get\_advertisedValue, YVSource 1418  
 get\_advertisedValue, YWakeUpMonitor 1448  
 get\_advertisedValue, YWakeUpSchedule 1483  
 get\_advertisedValue, YWatchdog 1521  
 get\_advertisedValue, YWireless 1565  
 get\_analogCalibration, YAnButton 80  
 get\_autoStart, YDataLogger 265  
 get\_autoStart, YWatchdog 1522  
 get\_averageValue, YDataRun 290  
 get\_averageValue, YDataStream 313  
 get\_averageValue, YMeasure 773  
 get\_baudRate, YHubPort 605  
 get\_beacon, YModule 789  
 get\_bearing, YRefFrame 1123  
 get\_bitDirection, YDigitalIO 334  
 get\_bitOpenDrain, YDigitalIO 335  
 get\_bitPolarity, YDigitalIO 336  
 get\_bitState, YDigitalIO 337  
 get\_blinking, YLed 669  
 get\_brightness, YDisplay 379  
 get\_calibratedValue, YAnButton 81  
 get\_calibrationMax, YAnButton 82  
 get\_calibrationMin, YAnButton 83  
 get\_callbackCredentials, YNetwork 831  
 get\_callbackEncoding, YNetwork 832  
 get\_callbackMaxDelay, YNetwork 833  
 get\_callbackMethod, YNetwork 834  
 get\_callbackMinDelay, YNetwork 835  
 get\_callbackUrl, YNetwork 836  
 get\_channel, YWireless 1566  
 get\_columnCount, YDataStream 314  
 get\_columnNames, YDataStream 315  
 get\_cosPhi, YPower 909  
 get\_countdown, YRelay 1154  
 get\_countdown, YWatchdog 1523  
 get\_currentRawValue, YAccelerometer 39  
 get\_currentRawValue, YCarbonDioxide 119  
 get\_currentRawValue, YCompass 187  
 get\_currentRawValue, YCurrent 227  
 get\_currentRawValue, YGenericSensor 510  
 get\_currentRawValue, YGyro 556  
 get\_currentRawValue, YHumidity 632  
 get\_currentRawValue, YLightSensor 699  
 get\_currentRawValue, YMagnetometer 738  
 get\_currentRawValue, YPower 910  
 get\_currentRawValue, YPressure 952  
 get\_currentRawValue, YQt 1052  
 get\_currentRawValue, YSensor 1190  
 get\_currentRawValue, YTemperature 1264  
 get\_currentRawValue, YTilt 1305  
 get\_currentRawValue, YVoc 1344  
 get\_currentRawValue, YVoltage 1383  
 get\_currentRunIndex, YDataLogger 266  
 get\_currentValue, YAccelerometer 40  
 get\_currentValue, YCarbonDioxide 120  
 get\_currentValue, YCompass 188  
 get\_currentValue, YCurrent 228  
 get\_currentValue, YGenericSensor 511  
 get\_currentValue, YGyro 557  
 get\_currentValue, YHumidity 633  
 get\_currentValue, YLightSensor 700  
 get\_currentValue, YMagnetometer 739  
 get\_currentValue, YPower 911  
 get\_currentValue, YPressure 953  
 get\_currentValue, YQt 1053  
 get\_currentValue, YSensor 1191  
 get\_currentValue, YTemperature 1265  
 get\_currentValue, YTilt 1306  
 get\_currentValue, YVoc 1345  
 get\_currentValue, YVoltage 1384  
 get\_data, YDataStream 316  
 get\_dataRows, YDataStream 317  
 get\_dataSamplesIntervalMs, YDataStream 318  
 get\_dataSets, YDataLogger 267  
 get\_dataStreams, YDataLogger 268  
 get\_dateTime, YRealTimeClock 1089  
 get\_detectedWlans, YWireless 1567  
 get\_discoverable, YNetwork 837  
 get\_display, YDisplayLayer 430  
 get\_displayHeight, YDisplay 380  
 get\_displayHeight, YDisplayLayer 431  
 get\_displayLayer, YDisplay 381  
 get\_displayType, YDisplay 382  
 get\_displayWidth, YDisplay 383  
 get\_displayWidth, YDisplayLayer 432  
 get\_duration, YDataRun 291  
 get\_duration, YDataStream 319  
 get\_dutyCycle, YPwmOutput 991  
 get\_dutyCycleAtPowerOn, YPwmOutput 992  
 get\_enabled, YDisplay 384  
 get\_enabled, YHubPort 606

get\_enabled, YPwmOutput 993  
get\_enabled, YServo 1228  
get\_enabledAtPowerOn, YPwmOutput 994  
get\_enabledAtPowerOn, YServo 1229  
get\_endTimeUTC, YDataSet 301  
get\_endTimeUTC, YMeasure 774  
get\_errorMessage, YAccelerometer 41  
get\_errorMessage, YAnButton 84  
get\_errorMessage, YCarbonDioxide 121  
get\_errorMessage, YColorLed 156  
get\_errorMessage, YCompass 189  
get\_errorMessage, YCurrent 229  
get\_errorMessage, YDataLogger 269  
get\_errorMessage, YDigitalIO 338  
get\_errorMessage, YDisplay 385  
get\_errorMessage, YDualPower 455  
get\_errorMessage, YFiles 482  
get\_errorMessage, YGenericSensor 512  
get\_errorMessage, YGyro 558  
get\_errorMessage, YHubPort 607  
get\_errorMessage, YHumidity 634  
get\_errorMessage, YLed 670  
get\_errorMessage, YLightSensor 701  
get\_errorMessage, YMagnetometer 740  
get\_errorMessage, YModule 790  
get\_errorMessage, YNetwork 838  
get\_errorMessage, YOsControl 884  
get\_errorMessage, YPower 912  
get\_errorMessage, YPressure 954  
get\_errorMessage, YPwmOutput 995  
get\_errorMessage, YPwmPowerSource 1027  
get\_errorMessage, YQt 1054  
get\_errorMessage, YRealTimeClock 1090  
get\_errorMessage, YRefFrame 1124  
get\_errorMessage, YRelay 1155  
get\_errorMessage, YSensor 1192  
get\_errorMessage, YServo 1230  
get\_errorMessage, YTemperature 1266  
get\_errorMessage, YTilt 1307  
get\_errorMessage, YVoc 1346  
get\_errorMessage, YVoltage 1385  
get\_errorMessage, YVSource 1419  
get\_errorMessage, YWakeUpMonitor 1449  
get\_errorMessage, YWakeUpSchedule 1484  
get\_errorMessage, YWatchdog 1524  
get\_errorMessage, YWireless 1568  
get\_errorType, YAccelerometer 42  
get\_errorType, YAnButton 85  
get\_errorType, YCarbonDioxide 122  
get\_errorType, YColorLed 157  
get\_errorType, YCompass 190  
get\_errorType, YCurrent 230  
get\_errorType, YDataLogger 270  
get\_errorType, YDigitalIO 339  
get\_errorType, YDisplay 386  
get\_errorType, YDualPower 456  
get\_errorType, YFiles 483  
get\_errorType, YGenericSensor 513  
get\_errorType, YGyro 559

get\_errorType, YHubPort 608  
get\_errorType, YHumidity 635  
get\_errorType, YLed 671  
get\_errorType, YLightSensor 702  
get\_errorType, YMagnetometer 741  
get\_errorType, YModule 791  
get\_errorType, YNetwork 839  
get\_errorType, YOsControl 885  
get\_errorType, YPower 913  
get\_errorType, YPressure 955  
get\_errorType, YPwmOutput 996  
get\_errorType, YPwmPowerSource 1028  
get\_errorType, YQt 1055  
get\_errorType, YRealTimeClock 1091  
get\_errorType, YRefFrame 1125  
get\_errorType, YRelay 1156  
get\_errorType, YSensor 1193  
get\_errorType, YServo 1231  
get\_errorType, YTemperature 1267  
get\_errorType, YTilt 1308  
get\_errorType, YVoc 1347  
get\_errorType, YVoltage 1386  
get\_errorType, YVSource 1420  
get\_errorType, YWakeUpMonitor 1450  
get\_errorType, YWakeUpSchedule 1485  
get\_errorType, YWatchdog 1525  
get\_errorType, YWireless 1569  
get\_extPowerFailure, YVSource 1421  
get\_extVoltage, YDualPower 457  
get\_failure, YVSource 1422  
get\_filesCount, YFiles 484  
get\_firmwareRelease, YModule 792  
get\_freeSpace, YFiles 485  
get\_frequency, YPwmOutput 997  
get\_friendlyName, YAccelerometer 43  
get\_friendlyName, YAnButton 86  
get\_friendlyName, YCarbonDioxide 123  
get\_friendlyName, YColorLed 158  
get\_friendlyName, YCompass 191  
get\_friendlyName, YCurrent 231  
get\_friendlyName, YDataLogger 271  
get\_friendlyName, YDigitalIO 340  
get\_friendlyName, YDisplay 387  
get\_friendlyName, YDualPower 458  
get\_friendlyName, YFiles 486  
get\_friendlyName, YGenericSensor 514  
get\_friendlyName, YGyro 560  
get\_friendlyName, YHubPort 609  
get\_friendlyName, YHumidity 636  
get\_friendlyName, YLed 672  
get\_friendlyName, YLightSensor 703  
get\_friendlyName, YMagnetometer 742  
get\_friendlyName, YNetwork 840  
get\_friendlyName, YOsControl 886  
get\_friendlyName, YPower 914  
get\_friendlyName, YPressure 956  
get\_friendlyName, YPwmOutput 998  
get\_friendlyName, YPwmPowerSource 1029  
get\_friendlyName, YQt 1056

get\_friendlyName, YRealTimeClock 1092  
get\_friendlyName, YRefFrame 1126  
get\_friendlyName, YRelay 1157  
get\_friendlyName, YSensor 1194  
get\_friendlyName, YServo 1232  
get\_friendlyName, YTemperature 1268  
get\_friendlyName, YTilt 1309  
get\_friendlyName, YVoc 1348  
get\_friendlyName, YVoltage 1387  
get\_friendlyName, YWakeUpMonitor 1451  
get\_friendlyName, YWakeUpSchedule 1486  
get\_friendlyName, YWatchdog 1526  
get\_friendlyName, YWireless 1570  
get\_functionDescriptor, YAccelerometer 44  
get\_functionDescriptor, YAnButton 87  
get\_functionDescriptor, YCarbonDioxide 124  
get\_functionDescriptor, YColorLed 159  
get\_functionDescriptor, YCompass 192  
get\_functionDescriptor, YCurrent 232  
get\_functionDescriptor, YDataLogger 272  
get\_functionDescriptor, YDigitalIO 341  
get\_functionDescriptor, YDisplay 388  
get\_functionDescriptor, YDualPower 459  
get\_functionDescriptor, YFiles 487  
get\_functionDescriptor, YGenericSensor 515  
get\_functionDescriptor, YGyro 561  
get\_functionDescriptor, YHubPort 610  
get\_functionDescriptor, YHumidity 637  
get\_functionDescriptor, YLed 673  
get\_functionDescriptor, YLightSensor 704  
get\_functionDescriptor, YMagnetometer 743  
get\_functionDescriptor, YNetwork 841  
get\_functionDescriptor, YOsControl 887  
get\_functionDescriptor, YPower 915  
get\_functionDescriptor, YPressure 957  
get\_functionDescriptor, YPwmOutput 999  
get\_functionDescriptor, YPwmPowerSource 1030  
get\_functionDescriptor, YQt 1057  
get\_functionDescriptor, YRealTimeClock 1093  
get\_functionDescriptor, YRefFrame 1127  
get\_functionDescriptor, YRelay 1158  
get\_functionDescriptor, YSensor 1195  
get\_functionDescriptor, YServo 1233  
get\_functionDescriptor, YTemperature 1269  
get\_functionDescriptor, YTilt 1310  
get\_functionDescriptor, YVoc 1349  
get\_functionDescriptor, YVoltage 1388  
get\_functionDescriptor, YVSource 1423  
get\_functionDescriptor, YWakeUpMonitor 1452  
get\_functionDescriptor, YWakeUpSchedule 1487  
get\_functionDescriptor, YWatchdog 1527  
get\_functionDescriptor, YWireless 1571  
get\_functionId, YAccelerometer 45  
get\_functionId, YAnButton 88  
get\_functionId, YCarbonDioxide 125  
get\_functionId, YColorLed 160  
get\_functionId, YCompass 193  
get\_functionId, YCurrent 233  
get\_functionId, YDataLogger 273

get\_functionId, YDataSet 302  
get\_functionId, YDigitalIO 342  
get\_functionId, YDisplay 389  
get\_functionId, YDualPower 460  
get\_functionId, YFiles 488  
get\_functionId, YGenericSensor 516  
get\_functionId, YGyro 562  
get\_functionId, YHubPort 611  
get\_functionId, YHumidity 638  
get\_functionId, YLed 674  
get\_functionId, YLightSensor 705  
get\_functionId, YMagnetometer 744  
get\_functionId, YNetwork 842  
get\_functionId, YOsControl 888  
get\_functionId, YPower 916  
get\_functionId, YPressure 958  
get\_functionId, YPwmOutput 1000  
get\_functionId, YPwmPowerSource 1031  
get\_functionId, YQt 1058  
get\_functionId, YRealTimeClock 1094  
get\_functionId, YRefFrame 1128  
get\_functionId, YRelay 1159  
get\_functionId, YSensor 1196  
get\_functionId, YServo 1234  
get\_functionId, YTemperature 1270  
get\_functionId, YTilt 1311  
get\_functionId, YVoc 1350  
get\_functionId, YVoltage 1389  
get\_functionId, YWakeUpMonitor 1453  
get\_functionId, YWakeUpSchedule 1488  
get\_functionId, YWatchdog 1528  
get\_functionId, YWireless 1572  
get\_hardwareId, YAccelerometer 46  
get\_hardwareId, YAnButton 89  
get\_hardwareId, YCarbonDioxide 126  
get\_hardwareId, YColorLed 161  
get\_hardwareId, YCompass 194  
get\_hardwareId, YCurrent 234  
get\_hardwareId, YDataLogger 274  
get\_hardwareId, YDataSet 303  
get\_hardwareId, YDigitalIO 343  
get\_hardwareId, YDisplay 390  
get\_hardwareId, YDualPower 461  
get\_hardwareId, YFiles 489  
get\_hardwareId, YGenericSensor 517  
get\_hardwareId, YGyro 563  
get\_hardwareId, YHubPort 612  
get\_hardwareId, YHumidity 639  
get\_hardwareId, YLed 675  
get\_hardwareId, YLightSensor 706  
get\_hardwareId, YMagnetometer 745  
get\_hardwareId, YModule 793  
get\_hardwareId, YNetwork 843  
get\_hardwareId, YOsControl 889  
get\_hardwareId, YPower 917  
get\_hardwareId, YPressure 959  
get\_hardwareId, YPwmOutput 1001  
get\_hardwareId, YPwmPowerSource 1032  
get\_hardwareId, YQt 1059

get\_hardwareId, YRealTimeClock 1095  
get\_hardwareId, YRefFrame 1129  
get\_hardwareId, YRelay 1160  
get\_hardwareId, YSensor 1197  
get\_hardwareId, YServo 1235  
get\_hardwareId, YTemperature 1271  
get\_hardwareId, YTilt 1312  
get\_hardwareId, YVoc 1351  
get\_hardwareId, YVoltage 1390  
get\_hardwareId, YWakeUpMonitor 1454  
get\_hardwareId, YWakeUpSchedule 1489  
get\_hardwareId, YWatchdog 1529  
get\_hardwareId, YWireless 1573  
get\_heading, YGyro 564  
get\_highestValue, YAccelerometer 47  
get\_highestValue, YCarbonDioxide 127  
get\_highestValue, YCompass 195  
get\_highestValue, YCurrent 235  
get\_highestValue, YGenericSensor 518  
get\_highestValue, YGyro 565  
get\_highestValue, YHumidity 640  
get\_highestValue, YLightSensor 707  
get\_highestValue, YMagnetometer 746  
get\_highestValue, YPower 918  
get\_highestValue, YPressure 960  
get\_highestValue, YQt 1060  
get\_highestValue, YSensor 1198  
get\_highestValue, YTemperature 1272  
get\_highestValue, YTilt 1313  
get\_highestValue, YVoc 1352  
get\_highestValue, YVoltage 1391  
get\_hours, YWakeUpSchedule 1490  
get\_hslColor, YColorLed 162  
get\_icon2d, YModule 794  
get\_ipAddress, YNetwork 844  
get\_isPressed, YAnButton 90  
get\_lastLogs, YModule 795  
get\_lastTimePressed, YAnButton 91  
get\_lastTimeReleased, YAnButton 92  
get\_layerCount, YDisplay 391  
get\_layerHeight, YDisplay 392  
get\_layerHeight, YDisplayLayer 433  
get\_layerWidth, YDisplay 393  
get\_layerWidth, YDisplayLayer 434  
get\_linkQuality, YWireless 1574  
get\_list, YFiles 490  
get\_logFrequency, YAccelerometer 48  
get\_logFrequency, YCarbonDioxide 128  
get\_logFrequency, YCompass 196  
get\_logFrequency, YCurrent 236  
get\_logFrequency, YGenericSensor 519  
get\_logFrequency, YGyro 566  
get\_logFrequency, YHumidity 641  
get\_logFrequency, YLightSensor 708  
get\_logFrequency, YMagnetometer 747  
get\_logFrequency, YPower 919  
get\_logFrequency, YPressure 961  
get\_logFrequency, YQt 1061  
get\_logFrequency, YSensor 1199  
get\_logFrequency, YTemperature 1273  
get\_logFrequency, YTilt 1314  
get\_logFrequency, YVoc 1353  
get\_logFrequency, YVoltage 1392  
get\_logicalName, YAccelerometer 49  
get\_logicalName, YAnButton 93  
get\_logicalName, YCarbonDioxide 129  
get\_logicalName, YColorLed 163  
get\_logicalName, YCompass 197  
get\_logicalName, YCurrent 237  
get\_logicalName, YDataLogger 275  
get\_logicalName, YDigitalIO 344  
get\_logicalName, YDisplay 394  
get\_logicalName, YDualPower 462  
get\_logicalName, YFiles 491  
get\_logicalName, YGenericSensor 520  
get\_logicalName, YGyro 567  
get\_logicalName, YHubPort 613  
get\_logicalName, YHumidity 642  
get\_logicalName, YLed 676  
get\_logicalName, YLightSensor 709  
get\_logicalName, YMagnetometer 748  
get\_logicalName, YModule 796  
get\_logicalName, YNetwork 845  
get\_logicalName, YOsControl 890  
get\_logicalName, YPower 920  
get\_logicalName, YPressure 962  
get\_logicalName, YPwmOutput 1002  
get\_logicalName, YPwmPowerSource 1033  
get\_logicalName, YQt 1062  
get\_logicalName, YRealTimeClock 1096  
get\_logicalName, YRefFrame 1130  
get\_logicalName, YRelay 1161  
get\_logicalName, YSensor 1200  
get\_logicalName, YServo 1236  
get\_logicalName, YTemperature 1274  
get\_logicalName, YTilt 1315  
get\_logicalName, YVoc 1354  
get\_logicalName, YVoltage 1393  
get\_logicalName, YVSource 1424  
get\_logicalName, YWakeUpMonitor 1455  
get\_logicalName, YWakeUpSchedule 1491  
get\_logicalName, YWatchdog 1530  
get\_logicalName, YWireless 1575  
get\_lowestValue, YAccelerometer 50  
get\_lowestValue, YCarbonDioxide 130  
get\_lowestValue, YCompass 198  
get\_lowestValue, YCurrent 238  
get\_lowestValue, YGenericSensor 521  
get\_lowestValue, YGyro 568  
get\_lowestValue, YHumidity 643  
get\_lowestValue, YLightSensor 710  
get\_lowestValue, YMagnetometer 749  
get\_lowestValue, YPower 921  
get\_lowestValue, YPressure 963  
get\_lowestValue, YQt 1063  
get\_lowestValue, YSensor 1201  
get\_lowestValue, YTemperature 1275  
get\_lowestValue, YTilt 1316

get\_lowestValue, YVoc 1355  
get\_lowestValue, YVoltage 1394  
get\_luminosity, YLed 677  
get\_luminosity, YModule 797  
get\_macAddress, YNetwork 846  
get\_magneticHeading, YCompass 199  
get\_maxTimeOnStateA, YRelay 1162  
get\_maxTimeOnStateA, YWatchdog 1531  
get\_maxTimeOnStateB, YRelay 1163  
get\_maxTimeOnStateB, YWatchdog 1532  
get\_maxValue, YDataRun 292  
get\_maxValue, YDataStream 320  
get\_maxValue, YMeasure 775  
get\_measureNames, YDataRun 293  
get\_measures, YDataSet 304  
get\_message, YWireless 1576  
get\_meter, YPower 922  
get\_meterTimer, YPower 923  
get\_minutes, YWakeUpSchedule 1492  
get\_minutesA, YWakeUpSchedule 1493  
get\_minutesB, YWakeUpSchedule 1494  
get\_minValue, YDataRun 294  
get\_minValue, YDataStream 321  
get\_minValue, YMeasure 776  
get\_module, YAccelerometer 51  
get\_module, YAnButton 94  
get\_module, YCarbonDioxide 131  
get\_module, YColorLed 164  
get\_module, YCompass 200  
get\_module, YCurrent 239  
get\_module, YDataLogger 276  
get\_module, YDigitalIO 345  
get\_module, YDisplay 395  
get\_module, YDualPower 463  
get\_module, YFiles 492  
get\_module, YGenericSensor 522  
get\_module, YGyro 569  
get\_module, YHubPort 614  
get\_module, YHumidity 644  
get\_module, YLed 678  
get\_module, YLightSensor 711  
get\_module, YMagnetometer 750  
get\_module, YNetwork 847  
get\_module, YOsControl 891  
get\_module, YPower 924  
get\_module, YPressure 964  
get\_module, YPwmOutput 1003  
get\_module, YPwmPowerSource 1034  
get\_module, YQt 1064  
get\_module, YRealTimeClock 1097  
get\_module, YRefFrame 1131  
get\_module, YRelay 1164  
get\_module, YSensor 1202  
get\_module, YServo 1237  
get\_module, YTemperature 1276  
get\_module, YTilt 1317  
get\_module, YVoc 1356  
get\_module, YVoltage 1395  
get\_module, YVSource 1425

get\_module, YWakeUpMonitor 1456  
get\_module, YWakeUpSchedule 1495  
get\_module, YWatchdog 1533  
get\_module, YWireless 1577  
get\_monthDays, YWakeUpSchedule 1496  
get\_months, YWakeUpSchedule 1497  
get\_mountOrientation, YRefFrame 1132  
get\_mountPosition, YRefFrame 1133  
get\_neutral, YServo 1238  
get\_nextOccurrence, YWakeUpSchedule 1498  
get\_nextWakeUp, YWakeUpMonitor 1457  
get\_orientation, YDisplay 396  
get\_output, YRelay 1165  
get\_output, YWatchdog 1534  
get\_outputVoltage, YDigitalIO 346  
get\_overCurrent, YVSource 1426  
get\_overHeat, YVSource 1427  
get\_overLoad, YVSource 1428  
get\_period, YPwmOutput 1004  
get\_persistentSettings, YModule 798  
get\_pitch, YGyro 570  
get\_poeCurrent, YNetwork 848  
get\_portDirection, YDigitalIO 347  
get\_portOpenDrain, YDigitalIO 348  
get\_portPolarity, YDigitalIO 349  
get\_portSize, YDigitalIO 350  
get\_portState, YDigitalIO 351  
get\_portState, YHubPort 615  
get\_position, YServo 1239  
get\_positionAtPowerOn, YServo 1240  
get\_power, YLed 679  
get\_powerControl, YDualPower 464  
get\_powerDuration, YWakeUpMonitor 1458  
get\_powerMode, YPwmPowerSource 1035  
get\_powerState, YDualPower 465  
get\_preview, YDataSet 305  
get\_primaryDNS, YNetwork 849  
get\_productId, YModule 799  
get\_productName, YModule 800  
get\_productRelease, YModule 801  
get\_progress, YDataSet 306  
get\_pulseCounter, YAnButton 95  
get\_pulseDuration, YPwmOutput 1005  
get\_pulseTimer, YAnButton 96  
get\_pulseTimer, YRelay 1166  
get\_pulseTimer, YWatchdog 1535  
get\_quaternionW, YGyro 571  
get\_quaternionX, YGyro 572  
get\_quaternionY, YGyro 573  
get\_quaternionZ, YGyro 574  
get\_range, YServo 1241  
get\_rawValue, YAnButton 97  
get\_readiness, YNetwork 850  
get\_rebootCountdown, YModule 802  
get\_recordedData, YAccelerometer 52  
get\_recordedData, YCarbonDioxide 132  
get\_recordedData, YCompass 201  
get\_recordedData, YCurrent 240  
get\_recordedData, YGenericSensor 523

get\_recordedData, YGyro 575  
get\_recordedData, YHumidity 645  
get\_recordedData, YLightSensor 712  
get\_recordedData, YMagnetometer 751  
get\_recordedData, YPower 925  
get\_recordedData, YPressure 965  
get\_recordedData, YQt 1065  
get\_recordedData, YSensor 1203  
get\_recordedData, YTemperature 1277  
get\_recordedData, YTilt 1318  
get\_recordedData, YVoc 1357  
get\_recordedData, YVoltage 1396  
get\_recording, YDataLogger 277  
get\_regulationFailure, YVSource 1429  
get\_reportFrequency, YAccelerometer 53  
get\_reportFrequency, YCarbonDioxide 133  
get\_reportFrequency, YCompass 202  
get\_reportFrequency, YCurrent 241  
get\_reportFrequency, YGenericSensor 524  
get\_reportFrequency, YGyro 576  
get\_reportFrequency, YHumidity 646  
get\_reportFrequency, YLightSensor 713  
get\_reportFrequency, YMagnetometer 752  
get\_reportFrequency, YPower 926  
get\_reportFrequency, YPressure 966  
get\_reportFrequency, YQt 1066  
get\_reportFrequency, YSensor 1204  
get\_reportFrequency, YTemperature 1278  
get\_reportFrequency, YTilt 1319  
get\_reportFrequency, YVoc 1358  
get\_reportFrequency, YVoltage 1397  
get\_resolution, YAccelerometer 54  
get\_resolution, YCarbonDioxide 134  
get\_resolution, YCompass 203  
get\_resolution, YCurrent 242  
get\_resolution, YGenericSensor 525  
get\_resolution, YGyro 577  
get\_resolution, YHumidity 647  
get\_resolution, YLightSensor 714  
get\_resolution, YMagnetometer 753  
get\_resolution, YPower 927  
get\_resolution, YPressure 967  
get\_resolution, YQt 1067  
get\_resolution, YSensor 1205  
get\_resolution, YTemperature 1279  
get\_resolution, YTilt 1320  
get\_resolution, YVoc 1359  
get\_resolution, YVoltage 1398  
get\_rgbColor, YColorLed 165  
get\_rgbColorAtPowerOn, YColorLed 166  
get\_roll, YGyro 578  
get\_router, YNetwork 851  
get\_rowCount, YDataStream 322  
get\_runIndex, YDataStream 323  
get\_running, YWatchdog 1536  
get\_secondaryDNS, YNetwork 852  
get\_security, YWireless 1578  
get\_sensitivity, YAnButton 98  
get\_sensorType, YTemperature 1280

get\_serialNumber, YModule 803  
get\_shutdownCountdown, YOsControl 892  
get\_signalRange, YGenericSensor 526  
get\_signalUnit, YGenericSensor 527  
get\_signalValue, YGenericSensor 528  
get\_sleepCountdown, YWakeUpMonitor 1459  
get\_ssid, YWireless 1579  
get\_startTime, YDataStream 324  
get\_startTimeUTC, YDataRun 295  
get\_startTimeUTC, YDataSet 307  
get\_startTimeUTC, YDataStream 325  
get\_startTimeUTC, YMeasure 777  
get\_startupSeq, YDisplay 397  
get\_state, YRelay 1167  
get\_state, YWatchdog 1537  
get\_stateAtPowerOn, YRelay 1168  
get\_stateAtPowerOn, YWatchdog 1538  
get\_subnetMask, YNetwork 853  
get\_summary, YDataSet 308  
get\_timeSet, YRealTimeClock 1098  
get\_timeUTC, YDataLogger 278  
get\_triggerDelay, YWatchdog 1539  
get\_triggerDuration, YWatchdog 1540  
get\_unit, YAccelerometer 55  
get\_unit, YCarbonDioxide 135  
get\_unit, YCompass 204  
get\_unit, YCurrent 243  
get\_unit, YDataSet 309  
get\_unit, YGenericSensor 529  
get\_unit, YGyro 579  
get\_unit, YHumidity 648  
get\_unit, YLightSensor 715  
get\_unit, YMagnetometer 754  
get\_unit, YPower 928  
get\_unit, YPressure 968  
get\_unit, YQt 1068  
get\_unit, YSensor 1206  
get\_unit, YTemperature 1281  
get\_unit, YTilt 1321  
get\_unit, YVoc 1360  
get\_unit, YVoltage 1399  
get\_unit, YVSource 1430  
get\_unixTime, YRealTimeClock 1099  
get\_upTime, YModule 804  
get\_usbBandwidth, YModule 805  
get\_usbCurrent, YModule 806  
get\_userData, YAccelerometer 56  
get\_userData, YAnButton 99  
get\_userData, YCarbonDioxide 136  
get\_userData, YColorLed 167  
get\_userData, YCompass 205  
get\_userData, YCurrent 244  
get\_userData, YDataLogger 279  
get\_userData, YDigitalIO 352  
get\_userData, YDisplay 398  
get\_userData, YDualPower 466  
get\_userData, YFiles 493  
get\_userData, YGenericSensor 530  
get\_userData, YGyro 580



- get\_userdata, YHubPort 616
- get\_userdata, YHumidity 649
- get\_userdata, YLed 680
- get\_userdata, YLightSensor 716
- get\_userdata, YMagnetometer 755
- get\_userdata, YModule 807
- get\_userdata, YNetwork 854
- get\_userdata, YOsControl 893
- get\_userdata, YPower 929
- get\_userdata, YPressure 969
- get\_userdata, YPwmOutput 1006
- get\_userdata, YPwmPowerSource 1036
- get\_userdata, YQt 1069
- get\_userdata, YRealTimeClock 1100
- get\_userdata, YRefFrame 1134
- get\_userdata, YRelay 1169
- get\_userdata, YSensor 1207
- get\_userdata, YServo 1242
- get\_userdata, YTemperature 1282
- get\_userdata, YTilt 1322
- get\_userdata, YVoc 1361
- get\_userdata, YVoltage 1400
- get\_userdata, YVSource 1431
- get\_userdata, YWakeUpMonitor 1460
- get\_userdata, YWakeUpSchedule 1499
- get\_userdata, YWatchdog 1541
- get\_userdata, YWireless 1580
- get\_userPassword, YNetwork 855
- get\_utcOffset, YRealTimeClock 1101
- get\_valueCount, YDataRun 296
- get\_valueInterval, YDataRun 297
- get\_valueRange, YGenericSensor 531
- get\_voltage, YVSource 1432
- get\_wakeUpReason, YWakeUpMonitor 1461
- get\_wakeUpState, YWakeUpMonitor 1462
- get\_weekDays, YWakeUpSchedule 1500
- get\_wwwWatchdogDelay, YNetwork 856
- get\_xValue, YAccelerometer 57
- get\_xValue, YGyro 581
- get\_xValue, YMagnetometer 756
- get\_yValue, YAccelerometer 58
- get\_yValue, YGyro 582
- get\_yValue, YMagnetometer 757
- get\_zValue, YAccelerometer 59
- get\_zValue, YGyro 583
- get\_zValue, YMagnetometer 758
- GetAPIVersion, YAPI 16
- GetTickCount, YAPI 17
- Gyro 549

## H

- HandleEvents, YAPI 18
- hide, YDisplayLayer 435
- Horloge 1084
- hslMove, YColorLed 168
- Humidity 625

## I

- InitAPI, YAPI 19
- Interface 32, 74, 112, 151, 180, 220, 259, 327, 371, 418, 450, 475, 503, 549, 600, 625, 664, 691, 731, 779, 822, 902, 945, 984, 1022, 1045, 1084, 1147, 1183, 1222, 1257, 1298, 1337, 1376, 1415, 1443, 1478, 1515, 1560
- Introduction 1
- isOnline, YAccelerometer 60
- isOnline, YAnButton 100
- isOnline, YCarbonDioxide 137
- isOnline, YColorLed 169
- isOnline, YCompass 206
- isOnline, YCurrent 245
- isOnline, YDataLogger 280
- isOnline, YDigitalIO 353
- isOnline, YDisplay 399
- isOnline, YDualPower 467
- isOnline, YFiles 494
- isOnline, YGenericSensor 532
- isOnline, YGyro 584
- isOnline, YHubPort 617
- isOnline, YHumidity 650
- isOnline, YLed 681
- isOnline, YLightSensor 717
- isOnline, YMagnetometer 759
- isOnline, YModule 808
- isOnline, YNetwork 857
- isOnline, YOsControl 894
- isOnline, YPower 930
- isOnline, YPressure 970
- isOnline, YPwmOutput 1007
- isOnline, YPwmPowerSource 1037
- isOnline, YQt 1070
- isOnline, YRealTimeClock 1102
- isOnline, YRefFrame 1135
- isOnline, YRelay 1170
- isOnline, YSensor 1208
- isOnline, YServo 1243
- isOnline, YTemperature 1283
- isOnline, YTilt 1323
- isOnline, YVoc 1362
- isOnline, YVoltage 1401
- isOnline, YVSource 1433
- isOnline, YWakeUpMonitor 1463
- isOnline, YWakeUpSchedule 1501
- isOnline, YWatchdog 1542
- isOnline, YWireless 1581

## J

- joinNetwork, YWireless 1582

## L

- Librairie 3
- LightSensor 691
- lineTo, YDisplayLayer 436

- load, YAccelerometer 61
- load, YAnButton 101
- load, YCarbonDioxide 138
- load, YColorLed 170
- load, YCompass 207
- load, YCurrent 246
- load, YDataLogger 281
- load, YDigitalIO 354
- load, YDisplay 400
- load, YDualPower 468
- load, YFiles 495
- load, YGenericSensor 533
- load, YGyro 585
- load, YHubPort 618
- load, YHumidity 651
- load, YLed 682
- load, YLightSensor 718
- load, YMagnetometer 760
- load, YModule 809
- load, YNetwork 858
- load, YOsControl 895
- load, YPower 931
- load, YPressure 971
- load, YPwmOutput 1008
- load, YPwmPowerSource 1038
- load, YQt 1071
- load, YRealTimeClock 1103
- load, YRefFrame 1136
- load, YRelay 1171
- load, YSensor 1209
- load, YServo 1244
- load, YTemperature 1284
- load, YTilt 1324
- load, YVoc 1363
- load, YVoltage 1402
- load, YVSource 1434
- load, YWakeUpMonitor 1464
- load, YWakeUpSchedule 1502
- load, YWatchdog 1543
- load, YWireless 1583
- loadCalibrationPoints, YAccelerometer 62
- loadCalibrationPoints, YCarbonDioxide 139
- loadCalibrationPoints, YCompass 208
- loadCalibrationPoints, YCurrent 247
- loadCalibrationPoints, YGenericSensor 534
- loadCalibrationPoints, YGyro 586
- loadCalibrationPoints, YHumidity 652
- loadCalibrationPoints, YLightSensor 719
- loadCalibrationPoints, YMagnetometer 761
- loadCalibrationPoints, YPower 932
- loadCalibrationPoints, YPressure 972
- loadCalibrationPoints, YQt 1072
- loadCalibrationPoints, YSensor 1210
- loadCalibrationPoints, YTemperature 1285
- loadCalibrationPoints, YTilt 1325
- loadCalibrationPoints, YVoc 1364
- loadCalibrationPoints, YVoltage 1403
- loadMore, YDataSet 310

## M

- Magnetometer 731
- Mesurée 773
- Mise 290
- Module 5, 779
- more3DCalibration, YRefFrame 1137
- move, YServo 1245
- moveTo, YDisplayLayer 437

## N

- Network 822
- newSequence, YDisplay 401
- nextAccelerometer, YAccelerometer 63
- nextAnButton, YAnButton 102
- nextCarbonDioxide, YCarbonDioxide 140
- nextColorLed, YColorLed 171
- nextCompass, YCompass 209
- nextCurrent, YCurrent 248
- nextDataLogger, YDataLogger 282
- nextDigitalIO, YDigitalIO 355
- nextDisplay, YDisplay 402
- nextDualPower, YDualPower 469
- nextFiles, YFiles 496
- nextGenericSensor, YGenericSensor 535
- nextGyro, YGyro 587
- nextHubPort, YHubPort 619
- nextHumidity, YHumidity 653
- nextLed, YLed 683
- nextLightSensor, YLightSensor 720
- nextMagnetometer, YMagnetometer 762
- nextModule, YModule 810
- nextNetwork, YNetwork 859
- nextOsControl, YOsControl 896
- nextPower, YPower 933
- nextPressure, YPressure 973
- nextPwmOutput, YPwmOutput 1009
- nextPwmPowerSource, YPwmPowerSource 1039
- nextQt, YQt 1073
- nextRealTimeClock, YRealTimeClock 1104
- nextRefFrame, YRefFrame 1138
- nextRelay, YRelay 1172
- nextSensor, YSensor 1211
- nextServo, YServo 1246
- nextTemperature, YTemperature 1286
- nextTilt, YTilt 1326
- nextVoc, YVoc 1365
- nextVoltage, YVoltage 1404
- nextVSource, YVSource 1435
- nextWakeUpMonitor, YWakeUpMonitor 1465
- nextWakeUpSchedule, YWakeUpSchedule 1503
- nextWatchdog, YWatchdog 1544
- nextWireless, YWireless 1584

## O

- Objets 418

## P

pauseSequence, YDisplay 403  
ping, YNetwork 860  
playSequence, YDisplay 404  
Port 600  
Power 902  
PreregisterHub, YAPI 20  
Pressure 945  
pulse, YDigitalIO 356  
pulse, YRelay 1173  
pulse, YVSource 1436  
pulse, YWatchdog 1545  
pulseDurationMove, YPwmOutput 1010  
PwmPowerSource 1022  
Python 3

## Q

Quaternion 1045

## R

Real 1084  
reboot, YModule 811  
Reference 10  
Référentiel 1111  
registerAnglesCallback, YGyro 588  
RegisterDeviceArrivalCallback, YAPI 21  
RegisterDeviceRemovalCallback, YAPI 22  
RegisterHub, YAPI 23  
RegisterHubDiscoveryCallback, YAPI 24  
registerLogCallback, YModule 812  
RegisterLogFunction, YAPI 25  
registerQuaternionCallback, YGyro 589  
registerTimedReportCallback, YAccelerometer 64  
registerTimedReportCallback, YCarbonDioxide 141  
registerTimedReportCallback, YCompass 210  
registerTimedReportCallback, YCurrent 249  
registerTimedReportCallback, YGenericSensor 536  
registerTimedReportCallback, YGyro 590  
registerTimedReportCallback, YHumidity 654  
registerTimedReportCallback, YLightSensor 721  
registerTimedReportCallback, YMagnetometer 763  
registerTimedReportCallback, YPower 934  
registerTimedReportCallback, YPressure 974  
registerTimedReportCallback, YQt 1074  
registerTimedReportCallback, YSensor 1212  
registerTimedReportCallback, YTemperature 1287  
registerTimedReportCallback, YTilt 1327  
registerTimedReportCallback, YVoc 1366  
registerTimedReportCallback, YVoltage 1405  
registerValueCallback, YAccelerometer 65  
registerValueCallback, YAnButton 103  
registerValueCallback, YCarbonDioxide 142

registerValueCallback, YColorLed 172  
registerValueCallback, YCompass 211  
registerValueCallback, YCurrent 250  
registerValueCallback, YDataLogger 283  
registerValueCallback, YDigitalIO 357  
registerValueCallback, YDisplay 405  
registerValueCallback, YDualPower 470  
registerValueCallback, YFiles 497  
registerValueCallback, YGenericSensor 537  
registerValueCallback, YGyro 591  
registerValueCallback, YHubPort 620  
registerValueCallback, YHumidity 655  
registerValueCallback, YLed 684  
registerValueCallback, YLightSensor 722  
registerValueCallback, YMagnetometer 764  
registerValueCallback, YNetwork 861  
registerValueCallback, YOsControl 897  
registerValueCallback, YPower 935  
registerValueCallback, YPressure 975  
registerValueCallback, YPwmOutput 1011  
registerValueCallback, YPwmPowerSource 1040  
registerValueCallback, YQt 1075  
registerValueCallback, YRealTimeClock 1105  
registerValueCallback, YRefFrame 1139  
registerValueCallback, YRelay 1174  
registerValueCallback, YSensor 1213  
registerValueCallback, YServo 1247  
registerValueCallback, YTemperature 1288  
registerValueCallback, YTilt 1328  
registerValueCallback, YVoc 1367  
registerValueCallback, YVoltage 1406  
registerValueCallback, YVSource 1437  
registerValueCallback, YWakeUpMonitor 1466  
registerValueCallback, YWakeUpSchedule 1504  
registerValueCallback, YWatchdog 1546  
registerValueCallback, YWireless 1585  
Relay 1147  
remove, YFiles 498  
reset, YDisplayLayer 438  
reset, YPower 936  
resetAll, YDisplay 406  
resetCounter, YAnButton 104  
resetSleepCountDown, YWakeUpMonitor 1467  
resetWatchdog, YWatchdog 1547  
revertFromFlash, YModule 813  
rgbMove, YColorLed 173

## S

save3DCalibration, YRefFrame 1140  
saveSequence, YDisplay 407  
saveToFlash, YModule 814  
SelectArchitecture, YAPI 26  
selectColorPen, YDisplayLayer 439  
selectEraser, YDisplayLayer 440  
selectFont, YDisplayLayer 441  
selectGrayPen, YDisplayLayer 442  
Senseur 1183  
Séquence 290, 300, 312  
Servo 1222

set\_adminPassword, YNetwork 862  
 set\_analogCalibration, YAnButton 105  
 set\_autoStart, YDataLogger 284  
 set\_autoStart, YWatchdog 1548  
 set\_beacon, YModule 815  
 set\_bearing, YRefFrame 1141  
 set\_bitDirection, YDigitalIO 358  
 set\_bitOpenDrain, YDigitalIO 359  
 set\_bitPolarity, YDigitalIO 360  
 set\_bitState, YDigitalIO 361  
 set\_blinking, YLed 685  
 set\_brightness, YDisplay 408  
 set\_calibrationMax, YAnButton 106  
 set\_calibrationMin, YAnButton 107  
 set\_callbackCredentials, YNetwork 863  
 set\_callbackEncoding, YNetwork 864  
 set\_callbackMaxDelay, YNetwork 865  
 set\_callbackMethod, YNetwork 866  
 set\_callbackMinDelay, YNetwork 867  
 set\_callbackUrl, YNetwork 868  
 set\_discoverable, YNetwork 869  
 set\_dutyCycle, YPwmOutput 1012  
 set\_dutyCycleAtPowerOn, YPwmOutput 1013  
 set\_enabled, YDisplay 409  
 set\_enabled, YHubPort 621  
 set\_enabled, YPwmOutput 1014  
 set\_enabled, YServo 1248  
 set\_enabledAtPowerOn, YPwmOutput 1015  
 set\_enabledAtPowerOn, YServo 1249  
 set\_frequency, YPwmOutput 1016  
 set\_highestValue, YAccelerometer 66  
 set\_highestValue, YCarbonDioxide 143  
 set\_highestValue, YCompass 212  
 set\_highestValue, YCurrent 251  
 set\_highestValue, YGenericSensor 538  
 set\_highestValue, YGyro 592  
 set\_highestValue, YHumidity 656  
 set\_highestValue, YLightSensor 723  
 set\_highestValue, YMagnetometer 765  
 set\_highestValue, YPower 937  
 set\_highestValue, YPressure 976  
 set\_highestValue, YQt 1076  
 set\_highestValue, YSensor 1214  
 set\_highestValue, YTemperature 1289  
 set\_highestValue, YTilt 1329  
 set\_highestValue, YVoc 1368  
 set\_highestValue, YVoltage 1407  
 set\_hours, YWakeUpSchedule 1505  
 set\_hslColor, YColorLed 174  
 set\_logFrequency, YAccelerometer 67  
 set\_logFrequency, YCarbonDioxide 144  
 set\_logFrequency, YCompass 213  
 set\_logFrequency, YCurrent 252  
 set\_logFrequency, YGenericSensor 539  
 set\_logFrequency, YGyro 593  
 set\_logFrequency, YHumidity 657  
 set\_logFrequency, YLightSensor 724  
 set\_logFrequency, YMagnetometer 766  
 set\_logFrequency, YPower 938  
 set\_logFrequency, YPressure 977  
 set\_logFrequency, YQt 1077  
 set\_logFrequency, YSensor 1215  
 set\_logFrequency, YTemperature 1290  
 set\_logFrequency, YTilt 1330  
 set\_logFrequency, YVoc 1369  
 set\_logFrequency, YVoltage 1408  
 set\_logicalName, YAccelerometer 68  
 set\_logicalName, YAnButton 108  
 set\_logicalName, YCarbonDioxide 145  
 set\_logicalName, YColorLed 175  
 set\_logicalName, YCompass 214  
 set\_logicalName, YCurrent 253  
 set\_logicalName, YDataLogger 285  
 set\_logicalName, YDigitalIO 362  
 set\_logicalName, YDisplay 410  
 set\_logicalName, YDualPower 471  
 set\_logicalName, YFiles 499  
 set\_logicalName, YGenericSensor 540  
 set\_logicalName, YGyro 594  
 set\_logicalName, YHubPort 622  
 set\_logicalName, YHumidity 658  
 set\_logicalName, YLed 686  
 set\_logicalName, YLightSensor 725  
 set\_logicalName, YMagnetometer 767  
 set\_logicalName, YModule 816  
 set\_logicalName, YNetwork 870  
 set\_logicalName, YOsControl 898  
 set\_logicalName, YPower 939  
 set\_logicalName, YPressure 978  
 set\_logicalName, YPwmOutput 1017  
 set\_logicalName, YPwmPowerSource 1041  
 set\_logicalName, YQt 1078  
 set\_logicalName, YRealTimeClock 1106  
 set\_logicalName, YRefFrame 1142  
 set\_logicalName, YRelay 1175  
 set\_logicalName, YSensor 1216  
 set\_logicalName, YServo 1250  
 set\_logicalName, YTemperature 1291  
 set\_logicalName, YTilt 1331  
 set\_logicalName, YVoc 1370  
 set\_logicalName, YVoltage 1409  
 set\_logicalName, YVSource 1438  
 set\_logicalName, YWakeUpMonitor 1468  
 set\_logicalName, YWakeUpSchedule 1506  
 set\_logicalName, YWatchdog 1549  
 set\_logicalName, YWireless 1586  
 set\_lowestValue, YAccelerometer 69  
 set\_lowestValue, YCarbonDioxide 146  
 set\_lowestValue, YCompass 215  
 set\_lowestValue, YCurrent 254  
 set\_lowestValue, YGenericSensor 541  
 set\_lowestValue, YGyro 595  
 set\_lowestValue, YHumidity 659  
 set\_lowestValue, YLightSensor 726  
 set\_lowestValue, YMagnetometer 768  
 set\_lowestValue, YPower 940  
 set\_lowestValue, YPressure 979  
 set\_lowestValue, YQt 1079

set\_lowestValue, YSensor 1217  
set\_lowestValue, YTemperature 1292  
set\_lowestValue, YTilt 1332  
set\_lowestValue, YVoc 1371  
set\_lowestValue, YVoltage 1410  
set\_luminosity, YLed 687  
set\_luminosity, YModule 817  
set\_maxTimeOnStateA, YRelay 1176  
set\_maxTimeOnStateA, YWatchdog 1550  
set\_maxTimeOnStateB, YRelay 1177  
set\_maxTimeOnStateB, YWatchdog 1551  
set\_minutes, YWakeUpSchedule 1507  
set\_minutesA, YWakeUpSchedule 1508  
set\_minutesB, YWakeUpSchedule 1509  
set\_monthDays, YWakeUpSchedule 1510  
set\_months, YWakeUpSchedule 1511  
set\_mountPosition, YRefFrame 1143  
set\_neutral, YServo 1251  
set\_nextWakeUp, YWakeUpMonitor 1469  
set\_orientation, YDisplay 411  
set\_output, YRelay 1178  
set\_output, YWatchdog 1552  
set\_outputVoltage, YDigitalIO 363  
set\_period, YPwmOutput 1018  
set\_portDirection, YDigitalIO 364  
set\_portOpenDrain, YDigitalIO 365  
set\_portPolarity, YDigitalIO 366  
set\_portState, YDigitalIO 367  
set\_position, YServo 1252  
set\_positionAtPowerOn, YServo 1253  
set\_power, YLed 688  
set\_powerControl, YDualPower 472  
set\_powerDuration, YWakeUpMonitor 1470  
set\_powerMode, YPwmPowerSource 1042  
set\_primaryDNS, YNetwork 871  
set\_pulseDuration, YPwmOutput 1019  
set\_range, YServo 1254  
set\_recording, YDataLogger 286  
set\_reportFrequency, YAccelerometer 70  
set\_reportFrequency, YCarbonDioxide 147  
set\_reportFrequency, YCompass 216  
set\_reportFrequency, YCurrent 255  
set\_reportFrequency, YGenericSensor 542  
set\_reportFrequency, YGyro 596  
set\_reportFrequency, YHumidity 660  
set\_reportFrequency, YLightSensor 727  
set\_reportFrequency, YMagnetometer 769  
set\_reportFrequency, YPower 941  
set\_reportFrequency, YPressure 980  
set\_reportFrequency, YQt 1080  
set\_reportFrequency, YSensor 1218  
set\_reportFrequency, YTemperature 1293  
set\_reportFrequency, YTilt 1333  
set\_reportFrequency, YVoc 1372  
set\_reportFrequency, YVoltage 1411  
set\_resolution, YAccelerometer 71  
set\_resolution, YCarbonDioxide 148  
set\_resolution, YCompass 217  
set\_resolution, YCurrent 256

set\_resolution, YGenericSensor 543  
set\_resolution, YGyro 597  
set\_resolution, YHumidity 661  
set\_resolution, YLightSensor 728  
set\_resolution, YMagnetometer 770  
set\_resolution, YPower 942  
set\_resolution, YPressure 981  
set\_resolution, YQt 1081  
set\_resolution, YSensor 1219  
set\_resolution, YTemperature 1294  
set\_resolution, YTilt 1334  
set\_resolution, YVoc 1373  
set\_resolution, YVoltage 1412  
set\_rgbColor, YColorLed 176  
set\_rgbColorAtPowerOn, YColorLed 177  
set\_running, YWatchdog 1553  
set\_secondaryDNS, YNetwork 872  
set\_sensitivity, YAnButton 109  
set\_sensorType, YTemperature 1295  
set\_signalRange, YGenericSensor 544  
set\_sleepCountdown, YWakeUpMonitor 1471  
set\_startupSeq, YDisplay 412  
set\_state, YRelay 1179  
set\_state, YWatchdog 1554  
set\_stateAtPowerOn, YRelay 1180  
set\_stateAtPowerOn, YWatchdog 1555  
set\_timeUTC, YDataLogger 287  
set\_triggerDelay, YWatchdog 1556  
set\_triggerDuration, YWatchdog 1557  
set\_unit, YGenericSensor 545  
set\_unixTime, YRealTimeClock 1107  
set\_usbBandwidth, YModule 818  
set\_userData, YAccelerometer 72  
set\_userData, YAnButton 110  
set\_userData, YCarbonDioxide 149  
set\_userData, YColorLed 178  
set\_userData, YCompass 218  
set\_userData, YCurrent 257  
set\_userData, YDataLogger 288  
set\_userData, YDigitalIO 368  
set\_userData, YDisplay 413  
set\_userData, YDualPower 473  
set\_userData, YFiles 500  
set\_userData, YGenericSensor 546  
set\_userData, YGyro 598  
set\_userData, YHubPort 623  
set\_userData, YHumidity 662  
set\_userData, YLed 689  
set\_userData, YLightSensor 729  
set\_userData, YMagnetometer 771  
set\_userData, YModule 819  
set\_userData, YNetwork 873  
set\_userData, YOsControl 899  
set\_userData, YPower 943  
set\_userData, YPressure 982  
set\_userData, YPwmOutput 1020  
set\_userData, YPwmPowerSource 1043  
set\_userData, YQt 1082  
set\_userData, YRealTimeClock 1108

set\_userdata, YRefFrame 1144  
set\_userdata, YRelay 1181  
set\_userdata, YSensor 1220  
set\_userdata, YServo 1255  
set\_userdata, YTemperature 1296  
set\_userdata, YTilt 1335  
set\_userdata, YVoc 1374  
set\_userdata, YVoltage 1413  
set\_userdata, YVSource 1439  
set\_userdata, YWakeUpMonitor 1472  
set\_userdata, YWakeUpSchedule 1512  
set\_userdata, YWatchdog 1558  
set\_userdata, YWireless 1587  
set\_userPassword, YNetwork 874  
set\_utcOffset, YRealTimeClock 1109  
set\_valueInterval, YDataRun 298  
set\_valueRange, YGenericSensor 547  
set\_voltage, YVSource 1440  
set\_weekDays, YWakeUpSchedule 1513  
set\_wwwWatchdogDelay, YNetwork 875  
setAntialiasingMode, YDisplayLayer 443  
setConsoleBackground, YDisplayLayer 444  
setConsoleMargins, YDisplayLayer 445  
setConsoleWordWrap, YDisplayLayer 446  
setLayerPosition, YDisplayLayer 447  
shutdown, YOsControl 900  
Sleep, YAPI 27  
sleep, YWakeUpMonitor 1473  
sleepFor, YWakeUpMonitor 1474  
sleepUntil, YWakeUpMonitor 1475  
Source 1415  
Sources 3  
start3DCalibration, YRefFrame 1145  
stopSequence, YDisplay 414  
swapLayerContent, YDisplay 415

## T

Temperature 1257  
Temps 1084  
Tension 1415  
Tilt 1298  
toggle\_bitState, YDigitalIO 369  
triggerFirmwareUpdate, YModule 820  
TriggerHubDiscovery, YAPI 28  
Type 1183

## U

unhide, YDisplayLayer 448  
UnregisterHub, YAPI 29  
UpdateDeviceList, YAPI 30  
upload, YDisplay 416  
upload, YFiles 501  
useDHCP, YNetwork 876  
useStaticIP, YNetwork 877

## V

Valeur 773

Voltage 1376  
voltageMove, YVSource 1441

## W

wakeUp, YWakeUpMonitor 1476  
WakeUpMonitor 1443  
WakeUpSchedule 1478  
Watchdog 1515  
Wireless 1560

## Y

YAccelerometer 34-72  
YAnButton 76-110  
YAPI 12-30  
YCarbonDioxide 114-149  
yCheckLogicalName 12  
YColorLed 152-178  
YCompass 182-218  
YCurrent 222-257  
YDataLogger 260-288  
YDataRun 290-298  
YDataSet 301-310  
YDataStream 313-325  
YDigitalIO 329-369  
yDisableExceptions 13  
YDisplay 373-416  
YDisplayLayer 419-448  
YDualPower 451-473  
yEnableExceptions 14  
YFiles 476-501  
yFindAccelerometer 34  
yFindAnButton 76  
yFindCarbonDioxide 114  
yFindColorLed 152  
yFindCompass 182  
yFindCurrent 222  
yFindDataLogger 260  
yFindDigitalIO 329  
yFindDisplay 373  
yFindDualPower 451  
yFindFiles 476  
yFindGenericSensor 505  
yFindGyro 551  
yFindHubPort 601  
yFindHumidity 627  
yFindLed 665  
yFindLightSensor 693  
yFindMagnetometer 733  
yFindModule 781  
yFindNetwork 825  
yFindOsControl 880  
yFindPower 904  
yFindPressure 947  
yFindPwmOutput 986  
yFindPwmPowerSource 1023  
yFindQt 1047  
yFindRealTimeClock 1085  
yFindRefFrame 1113

yFindRelay 1149	yFirstWatchdog 1518
yFindSensor 1185	yFirstWireless 1562
yFindServo 1224	yFreeAPI 15
yFindTemperature 1259	YGenericSensor 505-547
yFindTilt 1300	yGetAPIVersion 16
yFindVoc 1339	yGetTickCount 17
yFindVoltage 1378	YGyro 551-598
yFindVSource 1416	yHandleEvents 18
yFindWakeUpMonitor 1445	YHubPort 601-623
yFindWakeUpSchedule 1480	YHumidity 627-662
yFindWatchdog 1517	yInitAPI 19
yFindWireless 1561	YLed 665-689
yFirstAccelerometer 35	YLightSensor 693-729
yFirstAnButton 77	YMagnetometer 733-771
yFirstCarbonDioxide 115	YMeasure 773-777
yFirstColorLed 153	YModule 781-820
yFirstCompass 183	YNetwork 825-877
yFirstCurrent 223	Yocto-Demo 3
yFirstDataLogger 261	Yocto-hub 600
yFirstDigitalIO 330	YOsControl 880-900
yFirstDisplay 374	YPower 904-943
yFirstDualPower 452	yPreregisterHub 20
yFirstFiles 477	YPressure 947-982
yFirstGenericSensor 506	YPwmOutput 986-1020
yFirstGyro 552	YPwmPowerSource 1023-1043
yFirstHubPort 602	YQt 1047-1082
yFirstHumidity 628	YRealTimeClock 1085-1109
yFirstLed 666	YRefFrame 1113-1145
yFirstLightSensor 694	yRegisterDeviceArrivalCallback 21
yFirstMagnetometer 734	yRegisterDeviceRemovalCallback 22
yFirstModule 782	yRegisterHub 23
yFirstNetwork 826	yRegisterHubDiscoveryCallback 24
yFirstOsControl 881	yRegisterLogFunction 25
yFirstPower 905	YRelay 1149-1181
yFirstPressure 948	ySelectArchitecture 26
yFirstPwmOutput 987	YSensor 1185-1220
yFirstPwmPowerSource 1024	YServo 1224-1255
yFirstQt 1048	ySleep 27
yFirstRealTimeClock 1086	YTemperature 1259-1296
yFirstRefFrame 1114	YTilt 1300-1335
yFirstRelay 1150	yTriggerHubDiscovery 28
yFirstSensor 1186	yUnregisterHub 29
yFirstServo 1225	yUpdateDeviceList 30
yFirstTemperature 1260	YVoc 1339-1374
yFirstTilt 1301	YVoltage 1378-1413
yFirstVoc 1340	YVSource 1416-1441
yFirstVoltage 1379	YWakeUpMonitor 1445-1476
yFirstVSource 1417	YWakeUpSchedule 1480-1513
yFirstWakeUpMonitor 1446	YWatchdog 1517-1558
yFirstWakeUpSchedule 1481	YWireless 1561-1587