



Référence de l'API PYTHON

# Table des matières

<b>Introduction</b>	<b>3</b>
<b>Utilisation du Yocto—Demo en Python</b>	<b>4</b>
Fichiers sources	4
Librairie dynamique	4
Contrôle de la fonction Led	4
Contrôle de la partie module	6
Gestion des erreurs	8
<b>Reference</b>	<b>9</b>
Fonctions générales	9
Interface de la fonction AnButton	15
Interface de la fonction CarbonDioxide	25
Interface de la fonction ColorLed	34
Interface de la fonction Current	42
Interface de la fonction DataLogger	50
Séquence de données mise en forme	60
Séquence de données enregistrées	63
Interface de contrôle de l'alimentation	66
Interface d'un port de Yocto-hub	73
Interface de la fonction Humidity	80
Interface de la fonction Led	89
Interface de la fonction LightSensor	96
Interface de contrôle du module	105
Interface de la fonction Network	116
Interface de la fonction Pressure	129
Interface de la fonction Relay	138
Interface de la fonction Servo	145
Interface de la fonction Temperature	153
Interface de la fonction Voltage	163
Interface de la fonction Source de tension	171
Interface de la fonction Wireless	180
<b>Index</b>	<b>189</b>

# 1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie Python de Yoctopuce pour interfacier 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.

### 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.

```
[...]
```

---

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

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

```

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.

```

import os
import 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")

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 aux 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 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:

```
from yocto_api import *
```

#### 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.

##### `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.

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

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

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

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

#### **ySetTimeout(callback, ms\_timeout, optional\_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.

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

Configure la librairie Yoctopuce pour ne plus utiliser les modules connectés sur une machine préalablement enregistré 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()**

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()**

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()**

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).

---

Cette fonction est utilisée uniquement sous Android.

Avant d'appeler `yRegisterHub("usb")` il faut activer le port USB host du système. Cette fonction prend en argument un objet de la classe `android.content.Context` (ou d'une sous-classe). Il n'est pas nécessaire d'appeler cette fonction pour accéder aux modules à travers le réseau.

**Paramètres :**

**osContext** un objet de classe `android.content.Context` (ou une sous-classe)  
En cas d'erreur, déclenche une exception

---

## **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()**

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()**

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()

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()

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.RegisterDeviceArrivalCallback()

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` pour supprimer un callback déjà enregistré.

---

## YAPI.RegisterDeviceRemovalCallback()

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` pour supprimer un callback déjà enregistré.

---

## YAPI.RegisterHub()

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

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

Dans le cas d'une utilisation avec la passerelle VirtualHub, vous devez donner en paramètre l'adresse de la machine où tourne le VirtualHub (typiquement "`http://127.0.0.1:4444`", qui désigne la machine locale). Si vous utilisez un langage qui a un accès direct à USB, vous pouvez utiliser la pseudo-adresse "`usb`" à la place.

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 VirtualHub 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`

**Paramètres :**

**url** une chaîne de caractères contenant "`usb`" 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.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 déboguer le fonctionnement de l'API.

**Paramètres :**

**logfun** une procédure qui prend une chaîne de caractère en paramètre, ou `null` pour supprimer un callback déjà enregistré.

---

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

Les methodes `yDeviceArrival` et `yDeviceRemoval` seront appelées pendant l'exécution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

**Paramètres :**

**object** un objet qui soit se conformer au procol `YAPIDelegate`, ou `nil` pour supprimer un objet déjà enregistré.

---

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

Cette fonction se comporte plus ou moins comme la fonction Javascript `setTimeout`, mais durant le temps d'attente, elle va appeler `yHandleEvents` et `yUpdateDeviceList` périodiquement pour maintenir l'API à jour avec les modules connectés.

**Paramètres :**

**callback** la fonction à appeler lorsque le temps d'attente est écoulé. Sous Microsoft Internet Explorer, le callback doit être spécifié sous forme d'une string à évaluer.

**ms\_timeout** un entier correspondant à la durée de l'attente, en millisecondes

**optional\_arguments** des arguments supplémentaires peuvent être fournis, pour être passés à la fonction de callback si nécessaire (pas supporté sous Microsoft Internet Explorer).

**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.Sleep()

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.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 l'URL racine du VirtualHub à ne plus utiliser.

---

---

## 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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

### Paramètres :

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et le code de retour (`YAPI_SUCCESS` si l'opération se déroule sans erreur).

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## 3.2. 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, indépendamment de sa résistance totale.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
from yocto_anbutton import *
```

### 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 la fonction.

### `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 la fonction.

### `anbutton→get_errorType()`

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

### `anbutton→get_functionDescriptor()`

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

### `anbutton→get_hardwareId()`

Retourne l'identifiant unique de la fonction.

### `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_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 255 inclus) 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 la fonction est joignable, sans déclencher d'erreur.



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

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

**anbutton→load(msValidity)**

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

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

Met en cache les valeurs courantes de la fonction, 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→set\_analogCalibration(newval)**

Enclenche ou déclenche le procédure de calibration.

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

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 255 inclus) 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`.

## **YAnButton.FindAnButton()**

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()**

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()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **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()**

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()**

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()**

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()**

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()**

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

---

## **anbutton.get\_errorType()**

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

**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 fonction

---

## **anbutton.get\_anbuttonDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**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()**

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()**

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()**

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()**

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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**  
**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`  
**context** contexte fourni par l'appelant, et qui sera passé tel-qu'il est à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **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()**

Retourne la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclenchement de callbacks.

**def get\_sensitivity( )**

**Retourne :**  
un entier représentant la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclenchement de callbacks

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

---

## **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'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.

---

## **anbutton.isOnline()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**  
**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen  
**context** contexte fourni par l'appelant, et qui sera passé tel-qu'il est à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **anbutton.load()**

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.

---

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

---

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **anbutton.nextAnButton()**

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()**

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

### **anbutton.set\_analogCalibration()**

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()**

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()**

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()**

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'opération se déroule sans erreur.

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

---

### **anbutton.set\_sensitivity()**

Modifie la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclenchement de callbacks.

---



```
def set_sensitivity( newval)
```

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 255 inclus) 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()**

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 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:

```
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 la fonction.

#### **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 mesurée actuelle.

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

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

<b>carbondioxide→get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>carbondioxide→get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>carbondioxide→get_hardwareId()</b>	Retourne l'identifiant unique de la fonction.
<b>carbondioxide→get_highestValue()</b>	Retourne la valeur maximale observée.
<b>carbondioxide→get_logicalName()</b>	Retourne le nom logique du capteur de CO2.
<b>carbondioxide→get_lowestValue()</b>	Retourne la valeur minimale observée.
<b>carbondioxide→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>carbondioxide→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>carbondioxide→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>carbondioxide→get_unit()</b>	Retourne l'unité dans laquelle la valeur mesurée est exprimée.
<b>carbondioxide→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>carbondioxide→isOnline()</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>carbondioxide→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>carbondioxide→load(msValidity)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>carbondioxide→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>carbondioxide→nextCarbonDioxide()</b>	Continue l'énumération des capteurs de CO2 commencée à l'aide de yFirstCarbonDioxide().
<b>carbondioxide→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>carbondioxide→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.
<b>carbondioxide→set_logicalName(newval)</b>	Modifie le nom logique du capteur de CO2.
<b>carbondioxide→set_lowestValue(newval)</b>	Modifie la mémoire de valeur minimale observée.
<b>carbondioxide→set_userData(data)</b>	

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`.

---

## YCarbonDioxide.FindCarbonDioxide()

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()

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 à le premier capteur de CO2 accessible en ligne, ou `null` si il n'y a pas de capteurs de CO2 disponibles.

---

## carbondioxide.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.

---

**carbondioxide.describe()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

**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()**

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()**

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.

---

**carbondioxide.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

---

### **carbondioxide.get\_errorType()**

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

**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 fonction

---

### **carbondioxide.get\_carbondioxideDescriptor()**

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

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

### **carbondioxide.get\_highestValue()**

Retourne la valeur maximale observée.

**def get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

### **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()**

Retourne la valeur minimale observée.

**def get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

### **carbondioxide.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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **carbondioxide.get\_resolution()**

Retourne la résolution des valeurs mesurées.

**def get\_resolution( )**

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**def get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

## carbondioxide.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.

---

## carbondioxide.isOnline()

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## carbondioxide.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **carbondioxide.nextCarbonDioxide()**

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **carbondioxide.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.

---

## **carbondioxide.set\_logicalName()**

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'opération se déroule sans erreur.

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

---

## **carbondioxide.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.

---

## **carbondioxide.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

## 3.4. 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:

```
from yocto_colorled import *
```

### 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 la fonction.

#### **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 fonction.

#### **colorled→get\_errorType()**

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

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

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

#### **colorled→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

#### **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 fonction est joignable, sans déclencher d'erreur.

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

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

**colorled→load(msValidity)**

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

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

Met en cache les valeurs courantes de la fonction, 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`.

## YColorLed.FindColorLed()

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()**

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()**

Retourne un court texte décrivant la fonction.

def `describe()`

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

**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()**

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

---

**colorled.get\_errorType()**

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

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 fonction

---

### **colorled.get\_colorledDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

### **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()**

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()**

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`

---

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

---

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **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()**

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()**

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.

---

## **colorled.hsIMove()**

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

**def hsIMove( 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()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **colorled.load()**

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.

---

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

---

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **colorled.nextColorLed()**

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()**

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **colorled.rgbMove()**

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()**

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()**

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'opération se déroule sans erreur.

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

---

### **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()**

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.

---

### colored.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

## 3.5. 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:

```
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 la fonction.

**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 mesurée actuelle.

**current→get\_errorMessage()**

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

**current→get\_errorType()**

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

**current→get\_functionDescriptor()**

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

<b>current→get_hardwareId()</b>	Retourne l'identifiant unique de la fonction.
<b>current→get_highestValue()</b>	Retourne la valeur maximale observée.
<b>current→get_logicalName()</b>	Retourne le nom logique du capteur de courant.
<b>current→get_lowestValue()</b>	Retourne la valeur minimale observée.
<b>current→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>current→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>current→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>current→get_unit()</b>	Retourne l'unité dans laquelle la valeur mesurée est exprimée.
<b>current→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>current→isOnline()</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>current→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>current→load(msValidity)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>current→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>current→nextCurrent()</b>	Continue l'énumération des capteurs de courant commencée à l'aide de yFirstCurrent().
<b>current→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>current→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.
<b>current→set_logicalName(newval)</b>	Modifie le nom logique du capteur de courant.
<b>current→set_lowestValue(newval)</b>	Modifie la mémoire de valeur minimale observée.
<b>current→set_userData(data)</b>	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.

## YCurrent.FindCurrent()

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()**

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 à le premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

---

## **current.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.

---

## **current.describe()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**  
une chaîne de caractères décrivant la fonction

---

### **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()**

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()**

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`.

---

### **current.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

---

### **current.get\_errorType()**

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

---

`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 fonction

---

**current.get\_currentDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**current.get\_highestValue()**

Retourne la valeur maximale observée.

`def get_highestValue( )`

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

**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()**

Retourne la valeur minimale observée.

`def get_lowestValue( )`

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

## **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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**  
**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`  
**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **current.get\_resolution()**

Retourne la résolution des valeurs mesurées.

**def get\_resolution( )**

La résolution correspond à la précision de la représentation numérique 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()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**def get\_unit( )**

**Retourne :**  
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

---

## current.getUserData()

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

**def** `getUserData()` ( )

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.

---

## current.isOnline()

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**  
**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen  
**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## current.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **current.nextCurrent()**

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.
- 

### **current.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.

---

### **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'opération se déroule sans erreur.

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

---

### **current.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.

---

### **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

## **3.6. 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 bibliothèque de programmation Yoctopuce permet de contrôler le fonctionnement de l'enregistreur de données interne. Dans la mesure où les capteurs n'ont pas de pile intégrée, ils ne contiennent pas de référence de temps absolue. C'est pourquoi les mesures sont simplement indexées par le numéro de Run (période continue de fonctionnement lors d'une mise sous tension), et à l'intervalle de temps depuis le début du Run. Il est par contre possible

d'indiquer par logiciel à l'enregistreur de données l'heure UTC à un moment donnée, afin qu'il en tienne compte jusqu'à la prochaine mise hors tension.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
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 la fonction.

#### **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\_dataRun(runIdx)**

Retourne un objet YDataRun contenant toutes les données mesurées pour une période d'enclenchement du module donnée (un Run).

#### **datalogger→get\_dataStreams(v)**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur.

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

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

#### **datalogger→get\_errorType()**

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

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

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

#### **datalogger→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

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

Retourne le nom logique de l'enregistreur de données.

#### **datalogger→get\_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

#### **datalogger→get\_module()**

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

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

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

#### **datalogger→get\_oldestRunIndex()**

Retourne le numéro du Run le plus ancien pour lequel la mémoire non-volatile contient encore des données.

**`datalogger→get_recording()`**

Retourne l'état d'activation de l'enregistreur de données.

**`datalogger→get_timeUTC()`**

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

**`datalogger→get_userData()`**

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

**`datalogger→isOnline()`**

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

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

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

**`datalogger→load(msValidity)`**

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

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

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

**`datalogger→nextDataLogger()`**

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

**`datalogger→registerValueCallback(callback)`**

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

**`datalogger→set_autoStart(newval)`**

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

**`datalogger→set_logicalName(newval)`**

Modifie le nom logique de l'enregistreur de données.

**`datalogger→set_recording(newval)`**

Modifie l'état d'activation de l'enregistreur de données.

**`datalogger→set_timeUTC(newval)`**

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

**`datalogger→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`.

## **`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()`**

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 à le premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

---

## **`datalogger.describe()`**

Retourne un court texte décrivant la fonction.

**def `describe()`**

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **`datalogger.forgetAllDataStreams()`**

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()`**

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()**

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()**

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\_dataRun()**

Retourne un objet `YDataRun` contenant toutes les données mesurées pour une période d'enclenchement du module donnée (un Run).

**def get\_dataRun( runIdx)**

Cet objet pourra être utilisé pour récupérer les mesures (valeur min, valeur moyenne et valeur max) avec la granularité désirée.

**Paramètres :**

**runIdx** l'index du Run désiré

**Retourne :**

un objet `YDataRun`

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

---

## **datalogger.get\_dataStreams()**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur.

**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.

**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()**

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

---

## **datalogger.get\_errorType()**

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

**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 fonction

---

## **datalogger.get\_dataloggerDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

## **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`.

---

Retourne les noms des valeurs mesurées par l'enregistreur de données.

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.

---

### **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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**  
**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`  
**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **datalogger.get\_oldestRunIndex()**

Retourne le numéro du Run le plus ancien pour lequel la mémoire non-volatile contient encore des données.

**def get\_oldestRunIndex( )**

**Retourne :**  
un entier représentant le numéro du Run le plus ancien pour lequel la mémoire non-volatile contient encore des données

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

---

### **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()`**

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()`**

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.

---

### **`datalogger.isOnline()`**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**`callback`** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**`context`** contexte fourni par l'appelant, et qui sera passé tel-qu'il est à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **datalogger.load()**

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

### **Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### **Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **datalogger.nextDataLogger()**

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()**

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()**

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()**

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'opération se déroule sans erreur.

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

---

## **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()**

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()**

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

## **3.7. 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:

```
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()**

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()**

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()**

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()**

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()**

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`.

---

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()**

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()**

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()**

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

### 3.8. Séquence de données enregistrées

Les objets `DataStream` représentent des séquences de mesures enregistrées. Ils sont retournés par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
from yocto_datalogger import *
```

#### Méthodes des objets `YDataStream`

##### `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_dataSamplesInterval()`

Retourne le nombre de secondes entre chaque mesure de la 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 nombre de secondes entre le début du Run (mise sous tension du module) et le début de la séquence de données.

##### `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_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()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Retourne :**  
un entier positif correspondant au nombre de colonnes.

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

---

## **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 d'archivage résumant des séquences, un suffixe est ajouté à l'identifiant du capteur: `_min` pour la valeur minimale, `_avg` pour la valeur moyenne et `_max` pour la valeur maximale.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Retourne :**  
une liste de chaîne de caractères.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

---

## **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()**

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\_dataSamplesInterval()**

Retourne le nombre de secondes entre chaque mesure de la séquence.



**def `get_dataSamplesInterval()`**

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la création de séquences d'archive synthétisant de plus longue période peut produire des séquences plus espacées.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

**Retourne :**

un entier positif correspondant au nombre de secondes entre deux mesures consécutives.

---

**`datastream.getRowCount()`**

Retourne le nombre d'enregistrement contenus dans la séquence.

**def `getRowCount()`**

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Retourne :**

un entier positif correspondant au nombre d'enregistrements.

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

---

**`datastream.getRunIndex()`**

Retourne le numéro de Run de la séquence de données.

**def `getRunIndex()`**

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

**Retourne :**

un entier positif correspondant au numéro du Run

---

**`datastream.getStartTime()`**

Retourne le nombre de secondes entre le début du Run (mise sous tension du module) et le début de la séquence de données.

**def `getStartTime()`**

Si vous désirez obtenir l'heure absolue du début de la séquence, utilisez `get_startTimeUTC()`.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le début du Run et le début de la séquence enregistrée.

---

**`datastream.getStartTimeUTC()`**

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 `getStartTimeUTC()`**

Si l'heure UTC n'était pas configurée dans l'enregistreur de données au début de la séquence, cette méthode retourne 0.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début de la séquence enregistrée.

## 3.9. 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:

```
from yocto_dualpower import *
```

### 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 la fonction.

**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 de la fonction.

**dualpower→get\_errorType()**

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

**dualpower→get\_extVoltage()**

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

**dualpower→get\_functionDescriptor()**

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

**dualpower→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

**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 la fonction est joignable, sans déclencher d'erreur.

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

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

**dualpower→load(msValidity)**

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

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

Met en cache les valeurs courantes de la fonction, 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.

## YDualPower.FindDualPower()

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()**

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 à le premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

---

## **dualpower.describe()**

Retourne un court texte décrivant la fonction.

**def describe( )**

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **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()**

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

---

## **dualpower.get\_errorType()**

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

**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 fonction

---

### **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\_dualpowerDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

### **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()**

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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **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()**

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()**

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()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **dualpower.load()**

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **dualpower.nextDualPower()**

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()**

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **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'opération se déroule sans erreur.

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

---

## **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()

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

## 3.10. Interface d'un port de Yocto-hub

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
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 la fonction.

#### 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 de la fonction.

#### hubport→get\_errorType()

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

#### hubport→get\_functionDescriptor()

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

#### hubport→get\_hardwareId()

Retourne l'identifiant unique de la fonction.

<b>hubport→get_logicalName()</b>	Retourne le nom logique du port de Yocto-hub, qui est toujours le numéro de série du module qui y est connecté.
<b>hubport→get_module()</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>hubport→get_module_async(callback, context)</b>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b>hubport→get_portState()</b>	Retourne l'état actuel du port de Yocto-hub.
<b>hubport→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>hubport→isOnline()</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>hubport→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>hubport→load(msValidity)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>hubport→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>hubport→nextHubPort()</b>	Continue l'énumération des port de Yocto-hub commencée à l'aide de <code>yFirstHubPort()</code> .
<b>hubport→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>hubport→set_enabled(newval)</b>	Modifie le mode d'activation du port du Yocto-hub.
<b>hubport→set_logicalName(newval)</b>	Il n'est pas possible de configurer le nom logique d'un port de Yocto-hub.
<b>hubport→set_userData(data)</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> .

## 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()**

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 à le premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

---

## **hubport.describe()**

Retourne un court texte décrivant la fonction.

**def describe( )**

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **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()**

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()**

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()`**

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

---

**`hubport.get_errorType()`**

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

**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 fonction

---

**`hubport.get_hubportDescriptor()`**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

---

## hubport.get\_logicalName()

Retourne le nom logique du port de Yocto-hub, qui est toujours le numéro de série du module qui y est connecté.

**def get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du port de Yocto-hub, qui est toujours le numéro de série du module qui y est connecté

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

---

## 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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'il est à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## hubport.get\_portState()

Retourne l'état actuel du port de Yocto-hub.

**def get\_portState( )**

**Retourne :**

une valeur parmi `Y_PORTSTATE_OFF`, `Y_PORTSTATE_ON` et `Y_PORTSTATE_RUN` représentant l'état actuel du port de Yocto-hub

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

---

## hubport.get\_userData()

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

**def `getUserData()`**

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.

---

## **hubport.isOnline()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**  
**`callback`** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen  
**`context`** contexte fourni par l'appelant, et qui sera passé tel-qu'il est à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **hubport.load()**

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## hubport.nextHubPort()

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()

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## 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()

Il n'est pas possible de configurer le nom logique d'un port de Yocto-hub.

def set\_logicalName( newval)

Son nom est automatiquement configuré comme le numéro de série du module qui y est connecté.

**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.

---

## 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

## 3.11. 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:

```
from yocto_humidity import *
```

### 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 la fonction.

**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).
<b>humidity→get_currentValue()</b>	Retourne la valeur mesurée actuelle.
<b>humidity→get_errorMessage()</b>	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>humidity→get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>humidity→get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>humidity→get_hardwareId()</b>	Retourne l'identifiant unique de la fonction.
<b>humidity→get_highestValue()</b>	Retourne la valeur maximale observée.
<b>humidity→get_logicalName()</b>	Retourne le nom logique du capteur d'humidité.
<b>humidity→get_lowestValue()</b>	Retourne la valeur minimale observée.
<b>humidity→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>humidity→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>humidity→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>humidity→get_unit()</b>	Retourne l'unité dans laquelle la valeur mesurée est exprimée.
<b>humidity→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>humidity→isOnline()</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>humidity→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>humidity→load(msValidity)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>humidity→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>humidity→nextHumidity()</b>	Continue l'énumération des capteurs d'humidité commencée à l'aide de yFirstHumidity().
<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.
<b>humidity→set_logicalName(newval)</b>	

Modifie le nom logique du capteur d'humidité.

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

Modifie la mémoire de valeur minimale observée.

**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`.

## **YHumidity.FindHumidity()**

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()**

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 à le premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

## **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()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

**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()**

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()**

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.

---

**humidity.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

---

## **humidity.get\_errorType()**

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

**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 fonction

---

## **humidity.get\_humidityDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

## **humidity.get\_highestValue()**

Retourne la valeur maximale observée.

**def get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

## **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()`**

Retourne la valeur minimale observée.

**def `get_lowestValue( )`**

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

**`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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

**`humidity.get_resolution()`**

Retourne la résolution des valeurs mesurées.

**def `get_resolution( )`**

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**def get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

## humidity.getUserData()

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

**def getUserData( )**

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()

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## humidity.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## humidity.nextHumidity()

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **humidity.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.

---

## **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'opération se déroule sans erreur.

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

---

## **humidity.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.

---

## **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

## 3.12. 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:

```
from yocto_led import *
```

Fonction globales	
<b>yFindLed(func)</b>	Permet de retrouver une led d'après un identifiant donné.
<b>yFirstLed()</b>	Commence l'énumération des leds accessibles par la librairie.
Méthodes des objets YLed	
<b>led→describe()</b>	Retourne un court texte décrivant la fonction.
<b>led→get_advertisedValue()</b>	Retourne la valeur courante de la led (pas plus de 6 caractères).
<b>led→get_blinking()</b>	Retourne le mode de signalisation de la led.
<b>led→get_errorMessage()</b>	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>led→get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>led→get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>led→get_hardwareId()</b>	Retourne l'identifiant unique de la fonction.
<b>led→get_logicalName()</b>	Retourne le nom logique de la led.
<b>led→get_luminosity()</b>	Retourne l'intensité de la led en pour cent.
<b>led→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>led→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>led→get_power()</b>	Retourne l'état courant de la led.
<b>led→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>led→isOnline()</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>led→isOnline_async(callback, context)</b>	

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

#### **led**→**load**(**msValidity**)

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

#### **led**→**load\_async**(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de la fonction, 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`.

## **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()**

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()**

Retourne un court texte décrivant la fonction.

`def describe( )`

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

**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()**

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()**

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

---

**led.get\_errorType()**

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

`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 fonction

---

**led.get\_ledDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**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()**

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()**

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`

---

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **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.getUserData()**

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

**def getUserData( )**

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.

---

## **led.isOnline()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## led.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou **YAPI\_SUCCESS**)

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## led.nextLed()

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()**

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **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()**

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'opération se déroule sans erreur.

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

---

## **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()

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()

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

## 3.13. 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:

```
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 la fonction.

**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 valeur mesurée actuelle.

**lightsensor→get\_errorMessage()**

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

**lightsensor→get\_errorType()**

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

**lightsensor→get\_functionDescriptor()**

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

**lightsensor→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

**lightsensor→get\_highestValue()**

Retourne la valeur maximale observée.

**lightsensor→get\_logicalName()**

Retourne le nom logique du capteur de lumière.

**lightsensor→get\_lowestValue()**

Retourne la valeur minimale observée.

**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\_resolution()**

Retourne la résolution des valeurs mesurées.

**lightsensor→get\_unit()**

Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

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

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

**lightsensor→load(msValidity)**

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

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

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

**lightsensor→nextLightSensor()**

Continue l'énumération des capteurs de lumière commencée à l'aide de yFirstLightSensor().

**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.

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

Modifie le nom logique du capteur de lumière.

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

Modifie la mémoire de valeur minimale observée.

**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`.

## **YLightSensor.FindLightSensor()**

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()**

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 à le premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

## **lightsensor.calibrate()**

Modifie le paramètre de calibration spécifique du senseur 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.

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

**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()**

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()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **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()**

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()**

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`.

---

## **lightsensor.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

---

## **lightsensor.get\_errorType()**

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

**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 fonction

---

## **lightsensor.get\_lightsensorDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

### **lightsensor.get\_highestValue()**

Retourne la valeur maximale observée.

**def get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

### **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()**

Retourne la valeur minimale observée.

**def get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

### **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`

---

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

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

---

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **lightsensor.get\_resolution()**

Retourne la résolution des valeurs mesurées.

**def get\_resolution( )**

La résolution correspond à la précision de la représentation numérique 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()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**def get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

## **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()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## lightsensor.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

---

## lightsensor.nextLightSensor()

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## lightsensor.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.

---

## 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'opération se déroule sans erreur.

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



## lightsensor.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.

## 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

## 3.14. 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:

```
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→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.

<b>module→get_errorMessage()</b>	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.
<b>module→get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.
<b>module→get_firmwareRelease()</b>	Retourne la version du logiciel embarqué du module.
<b>module→get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>module→get_hardwareId()</b>	Retourne l'identifiant unique du module.
<b>module→get_icon2d()</b>	Retourne l'icone du module.
<b>module→get_logicalName()</b>	Retourne le nom logique du module.
<b>module→get_luminosity()</b>	Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).
<b>module→get_persistentSettings()</b>	Retourne l'état courant des réglages persistents du module.
<b>module→get_productId()</b>	Retourne l'identifiant USB du module, préprogrammé en usine.
<b>module→get_productName()</b>	Retourne le nom commercial du module, préprogrammé en usine.
<b>module→get_productRelease()</b>	Retourne le numéro de version matériel du module, préprogrammé en usine.
<b>module→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→get_serialNumber()</b>	Retourne le numéro de série du module, préprogrammé en usine.
<b>module→get_upTime()</b>	Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module
<b>module→get_usbBandwidth()</b>	Retourne le nombre d'interface USB utilisé par le module.
<b>module→get_usbCurrent()</b>	Retourne le courant consommé par le module sur le bus USB, en milliampères.
<b>module→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
<b>module→isOnline()</b>	Vérifie si le module est joignable, sans déclencher d'erreur.
<b>module→isOnline_async(callback, context)</b>	Vérifie si le module est joignable, sans déclencher d'erreur.
<b>module→load(msValidity)</b>	

<code>Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.</code>
<b>module</b> → <b>load_async</b> ( <b>msValidity</b> , <b>callback</b> , <b>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</b> ( <b>secBeforeReboot</b> ) Agende un simple redémarrage du module dans un nombre donné de secondes.
<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.
<b>module</b> → <b>set_beacon</b> ( <b>newval</b> ) Allume ou éteint la balise de localisation du module.
<b>module</b> → <b>set_logicalName</b> ( <b>newval</b> ) Change le nom logique du module.
<b>module</b> → <b>set_luminosity</b> ( <b>newval</b> ) Modifie la luminosité des leds informatives du module.
<b>module</b> → <b>set_usbBandwidth</b> ( <b>newval</b> ) Modifie le nombre d'interface USB utilisé par le module.
<b>module</b> → <b>set_userData</b> ( <b>data</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>module</b> → <b>triggerFirmwareUpdate</b> ( <b>secBeforeReboot</b> ) Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

## YModule.FindModule()

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()

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()**

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.functionCount()**

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()**

Retourne l'identifiant matériel de la *n*ième 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()**

Retourne le nom logique de la *n*ième 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()**

Retourne la valeur publiée par la *n*ième 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()**

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()**

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()**

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()**

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\_moduleDescriptor()

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`

---

Retourne l'identifiant unique du module.

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

**Retourne :**

une chaîne de caractères identifiant la fonction

---

Retourne l'icone du module.

L'icone est au format png et a une taille maximale de 1024 octets.

**Retourne :**

un buffer binaire contenant l'icone, au format png.

---

## 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()

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()

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.getProductId()**

Retourne l'identifiant USB du module, préprogrammé en usine.

**def getProductId( )**

**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.getProductName()**

Retourne le nom commercial du module, préprogrammé en usine.

**def getProductName( )**

**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.getProductRelease()**

Retourne le numéro de version matériel du module, préprogrammé en usine.

**def getProductRelease( )**

**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.getRebootCountdown()**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

**def getRebootCountdown( )**

**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.getSerialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

**def getSerialNumber( )**

**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()**

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()**

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()**

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.getUserData()**

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

**def getUserData( )**

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.

---

## **module.isOnline()**

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

---

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

---



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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## module.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## module.nextModule()

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()**

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.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()**

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()**

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()

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()

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()

Modifie le nombre d'interface USB utilisé par le module.

```
def set_usbBandwidth( newval)
```

**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()

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()

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.

## 3.15. 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:

```
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 la fonction.

**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 si il a été configuré, ou sinon une chaîne vide.

**network→get\_callbackMaxDelay**()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

**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\_errorMessage**()

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

**network→get\_errorType**()

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

**network→get\_functionDescriptor()**

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

**network→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

**network→get\_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

**network→get\_logicalName()**

Retourne le nom logique de l'interface réseau, qui correspond au nom réseau du module.

**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\_primaryDNS()**

Retourne l'adresse IP du serveur de nom 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 nom 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→isOnline()**

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

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

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

**network→load(msValidity)**

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

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

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

**network→nextNetwork()**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

**network→registerValueCallback(callback)**

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

<b>network→set_adminPassword(newval)</b>
Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.
<b>network→set_callbackCredentials(newval)</b>
Modifie le laisser-passer pour se connecter à l'adresse de callback.
<b>network→set_callbackMaxDelay(newval)</b>
Modifie l'attente maximale entre deux notifications par callback, en secondes.
<b>network→set_callbackMinDelay(newval)</b>
Modifie l'attente minimale entre deux notifications par callback, en secondes.
<b>network→set_callbackUrl(newval)</b>
Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.
<b>network→set_logicalName(newval)</b>
Modifie le nom logique de l'interface réseau, qui correspond au nom réseau du module.
<b>network→set_primaryDNS(newval)</b>
Modifie l'adresse IP du serveur de nom primaire que le module doit utiliser.
<b>network→set_secondaryDNS(newval)</b>
Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.
<b>network→set_userData(data)</b>
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.
<b>network→set_userPassword(newval)</b>
Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.
<b>network→useDHCP(fallbackIpAddress, fallbackSubnetMaskLen, fallbackRouter)</b>
Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.
<b>network→useStaticIP(ipAddress, subnetMaskLen, router)</b>
Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

## 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()**

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()**

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()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**  
une chaîne de caractères décrivant la fonction

---

## **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()**

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()**

Retourne une version hashée du laisser-passer pour le callback de notification si 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 si il a été configuré, ou sinon une chaîne vide

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

---

**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\_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()**

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\_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

---

**network.getErrorType()**

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

**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 fonction

---

**network.get\_networkDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction. En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**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()**

Retourne le nom logique de l'interface réseau, qui correspond au nom réseau du module.

---

**def get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau, qui correspond au nom réseau du module

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

---

## **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()**

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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **network.get\_primaryDNS()**

Retourne l'adresse IP du serveur de nom primaire que le module doit utiliser.

**def get\_primaryDNS( )**

**Retourne :**

---

une chaîne de caractères représentant l'adresse IP du serveur de nom primaire que le module doit utiliser

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

---

## **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. 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_LINK_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()**

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()**

Retourne l'adresse IP du serveur de nom secondaire que le module doit utiliser.

**def get\_secondaryDNS( )**

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

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

---

## **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.getUserData()**

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

**def getUserData( )**

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.getUserPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

**def getUserPassword( )**

**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.isOnline()**

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

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'il est à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## network.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## network.nextNetwork()

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **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()**

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\_callbackMaxDelay()**

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\_callbackMinDelay()**

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()**

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\_logicalName()**

Modifie le nom logique de l'interface réseau, qui correspond au nom réseau 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 représentant le nom logique de l'interface réseau, qui correspond au nom réseau 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\_primaryDNS()**

Modifie l'adresse IP du serveur de nom 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 nom 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()

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()

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()

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.useDHCP()

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 :**

**fallbackIpAddr** adresse IP à utiliser si aucun serveur DHCP ne répond  
**fallbackSubnetMaskLen** longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.  
**fallbackRouter** 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()

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 :**

**ipAddress** adresse IP à utiliser par le module  
**subnetMaskLen** longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.  
**router** 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.

## 3.16. 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:

`from yocto_pressure import *`

### 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 la fonction.

**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 valeur mesurée actuelle.

**pressure→get\_errorMessage()**

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

**pressure→get\_errorType()**

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

**pressure→get\_functionDescriptor()**

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

**pressure→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

**pressure→get\_highestValue()**

Retourne la valeur maximale observée.

**pressure→get\_logicalName()**

Retourne le nom logique du capteur de pression.

**pressure→get\_lowestValue()**

Retourne la valeur minimale observée.

**pressure→get\_module()**

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

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

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

**pressure→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**pressure→get\_unit()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**pressure→get\_userData()**

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

**pressure→isOnline()**

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

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

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

**pressure→load(msValidity)**

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

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

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

**pressure→nextPressure()**

Continue l'énumération des capteurs de pression commencée à l'aide de yFirstPressure().

**pressure→registerValueCallback(callback)**

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

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

Modifie la mémoire de valeur maximale observée.

**pressure→set\_logicalName(newval)**

Modifie le nom logique du capteur de pression.

**pressure→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**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`.

## YPressure.FindPressure()

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()

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 à le premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

## pressure.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.

---

## **pressure.describe()**

Retourne un court texte décrivant la fonction.

**def describe()**

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **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()**

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()**

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`.

---

## **pressure.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

---

## **pressure.get\_errorType()**

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

**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 fonction

---

## **pressure.get\_pressureDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

## **pressure.get\_highestValue()**

Retourne la valeur maximale observée.

**def get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

## **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()**

Retourne la valeur minimale observée.

**def get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

## **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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **pressure.get\_resolution()**

Retourne la résolution des valeurs mesurées.

**def get\_resolution( )**

La résolution correspond à la précision de la représentation numérique 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()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

`def get_unit( )`

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

**pressure.getUserData()**

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

`def getUserData( )`

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.

---

**pressure.isOnline()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

---

## pressure.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

### Paramètres :

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## pressure.nextPressure()

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **pressure.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.

---

## **pressure.set\_logicalName()**

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'opération se déroule sans erreur.

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

---

## **pressure.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.

---

## **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

## 3.17. 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 (relais 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 préférez l'état par défaut opposé, vous pouvez simplement changer vos fils sur le bornier.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
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→describe()**

Retourne un court texte décrivant la fonction.

#### **relay→get\_advertisedValue()**

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

#### **relay→get\_errorMessage()**

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

#### **relay→get\_errorType()**

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

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

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

#### **relay→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

#### **relay→get\_logicalName()**

Retourne le nom logique du relais.

#### **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.

<b>relay→get_state()</b>	Retourne l'état du relais (A pour la position de repos, B pour l'état actif).
<b>relay→get_userdata()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.
<b>relay→isOnline()</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>relay→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<b>relay→load(msValidity)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>relay→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<b>relay→nextRelay()</b>	Continue l'énumération des relais commencée à l'aide de yFirstRelay().
<b>relay→pulse(ms_duration)</b>	Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).
<b>relay→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>relay→set_logicalName(newval)</b>	Modifie le nom logique du relais.
<b>relay→set_output(newval)</b>	Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.
<b>relay→set_state(newval)</b>	Modifie l'état du relais (A pour la position de repos, B pour l'état actif).
<b>relay→set_userdata(data)</b>	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.

## 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()`**

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 à le premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

---

## **`relay.describe()`**

Retourne un court texte décrivant la fonction.

**def `describe()`**

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **`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_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

---

## **`relay.get_errorType()`**

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

**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 fonction

---

## **relay.get\_relayDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

## **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\_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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui

n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **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()**

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()**

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\_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.

---

---

## relay.isOnline()

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## relay.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui

n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **relay.nextRelay()**

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()**

Commute le relais à l'état B (actif) pour une 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()**

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **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'opération se déroule sans erreur.

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

---

### **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()**

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\_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

## **3.18. 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:

from yocto\_servo import \*

## 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 la fonction.

### servo→get\_advertisedValue()

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

### servo→get\_errorMessage()

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

### servo→get\_errorType()

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

### servo→get\_functionDescriptor()

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

### servo→get\_hardwareId()

Retourne l'identifiant unique de la fonction.

### 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\_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 la fonction est joignable, sans déclencher d'erreur.

### servo→isOnline\_async(callback, context)

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

### servo→load(msValidity)

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

### servo→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes de la fonction, 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\_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\_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`.

## **YServo.FindServo()**

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()**

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 à le premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

---

## **servo.describe()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **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\_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

---

## **servo.get\_errorType()**

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

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 fonction

---

## **servo.get\_servoDescriptor()**

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`

---

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

### **`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()`**

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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **`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()`**

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_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.getUserData()`**

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

`def getUserData( )`

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()`**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

---

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**servo.load()**

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.

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**servo.move()**

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()**

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()**

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\_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'opération se déroule sans erreur.

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

---

### **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()**

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\_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()**

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 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:

```
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 la fonction.

#### `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 mesurée actuelle.

#### `temperature→get_errorMessage()`

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

#### `temperature→get_errorType()`

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

#### `temperature→get_functionDescriptor()`

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

#### `temperature→get_hardwareId()`

Retourne l'identifiant unique de la fonction.

#### `temperature→get_highestValue()`

Retourne la valeur maximale observée.

#### `temperature→get_logicalName()`

Retourne le nom logique du capteur de température.

#### `temperature→get_lowestValue()`

Retourne la valeur minimale observée.

#### `temperature→get_module()`

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

#### `temperature→get_module_async(callback, context)`

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

#### `temperature→get_resolution()`

Retourne la résolution des valeurs mesurées.

#### `temperature→get_sensorType()`

Retourne le type de capteur de température utilisé par le module

#### `temperature→get_unit()`

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

#### `temperature→get_userData()`

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

#### `temperature→isOnline()`

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

#### `temperature→isOnline_async(callback, context)`

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

**temperature→load(msValidity)**

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

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

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

**temperature→nextTemperature()**

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

**temperature→registerValueCallback(callback)**

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

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

Modifie la mémoire de valeur maximale observée.

**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\_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`.

## 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()

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 à le premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

---

**temperature.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.

---

**temperature.describe()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

**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()**

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()`**

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`.

---

**`temperature.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

---

**`temperature.get_errorType()`**

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

**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 fonction

---

**`temperature.get_temperatureDescriptor()`**

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`

---

Retourne l'identifiant unique de la fonction.

---

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

### **temperature.get\_highestValue()**

Retourne la valeur maximale observée.

**def get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

### **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()**

Retourne la valeur minimale observée.

**def get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

### **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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui

n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **temperature.get\_resolution()**

Retourne la résolution des valeurs mesurées.

**def get\_resolution( )**

La résolution correspond à la précision de la représentation numérique 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()**

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` et `Y_SENSORTYPE_TYPE_T` 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()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**def get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

## **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'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.

---

---

## temperature.isOnline()

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## temperature.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui



n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **temperature.nextTemperature()**

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **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\_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'opération se déroule sans erreur.

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

---

## **temperature.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.

---

## **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` et `Y_SENSORTYPE_TYPE_T`

**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()**

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

## 3.20. 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:

```
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 la fonction.

#### **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 mesurée actuelle.

#### **voltage→get\_errorMessage()**

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

#### **voltage→get\_errorType()**

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

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

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

#### **voltage→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

#### **voltage→get\_highestValue()**

Retourne la valeur maximale observée.

#### **voltage→get\_logicalName()**

Retourne le nom logique du capteur de tension.

#### **voltage→get\_lowestValue()**

Retourne la valeur minimale observée.

#### **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\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **voltage→get\_unit()**

Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

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

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

**voltage→load(msValidity)**

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

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

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

**voltage→nextVoltage()**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

**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.

**voltage→set\_logicalName(newval)**

Modifie le nom logique du capteur de tension.

**voltage→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**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`.

## YVoltage.FindVoltage()

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()

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 à le premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

---

## voltage.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.

---

## voltage.describe()

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## 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()**

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()**

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`.

---

## **voltage.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

---

## **voltage.get\_errorType()**

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

**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 fonction

---

## **voltage.get\_voltageDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**voltage.get\_highestValue()**

Retourne la valeur maximale observée.

**def get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée

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

---

**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()**

Retourne la valeur minimale observée.

**def get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée

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

---

**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`

---

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

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

---

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **voltage.get\_resolution()**

Retourne la résolution des valeurs mesurées.

### **def get\_resolution( )**

La résolution correspond à la précision de la représentation numérique 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()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

### **def get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

## **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()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## voltage.load()

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

---

## **voltage.nextVoltage()**

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.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **voltage.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.

---

## **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'opération se déroule sans erreur.

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

---

## **voltage.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.

---

## **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

## **3.21. 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:

```
from yocto_vsource import *
```

### **Fonction globales**

#### **yFindVSource(func)**

Permet de retrouver une source de tension d'après un identifiant donné.

#### **yFirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

### **Méthodes des objets YVSource**

#### **vsource→describe()**

Retourne un court texte décrivant la fonction.

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

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

#### **vsource→get\_errorMessage()**

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

#### **vsource→get\_errorType()**

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

#### **vsource→get\_extPowerFailure()**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

#### **vsource→get\_failure()**

Indique si le module est en condition d'erreur.

---

<b><code>vsource→get_functionDescriptor()</code></b>
Retourne un identifiant unique de type <code>YFUN_DESCR</code> correspondant à la fonction.
<b><code>vsource→get_hardwareId()</code></b>
Retourne l'identifiant unique de la fonction.
<b><code>vsource→get_logicalName()</code></b>
Retourne le nom logique de la source de tension.
<b><code>vsource→get_module()</code></b>
Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b><code>vsource→get_module_async(callback, context)</code></b>
Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<b><code>vsource→get_overCurrent()</code></b>
Rend <code>TRUE</code> si l'appareil connecté à la sortie du module consomme trop de courant.
<b><code>vsource→get_overHeat()</code></b>
Rend <code>TRUE</code> si le module est en surchauffe.
<b><code>vsource→get_overLoad()</code></b>
Rend <code>TRUE</code> si le module n'est pas capable de tenir la tension de sortie demandée.
<b><code>vsource→get_regulationFailure()</code></b>
Rend <code>TRUE</code> 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→reset()</code></b>
Réinitialise la sortie du module.
<b><code>vsource→set_logicalName(newval)</code></b>
Modifie le nom logique de la source de tension.

**vsource→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`.

**vsource→set\_voltage(newval)**

Règle la tension de sortie du module (en milliVolts).

**vsource→voltageMove(target, ms\_duration)**

Déclenche une variation constante de la sortie vers une valeur donnée.

**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.

**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.describe()**

Retourne un court texte décrivant la fonction.

**def describe( )**

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**

une chaîne de caractères décrivant la fonction

---

## **vsource.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.

---

## **vsource.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

---

## **vsource.get\_errorType()**

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

**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 fonction

---

## **vsource.get\_extPowerFailure()**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

**def get\_extPowerFailure( )**

**Retourne :**

soit Y\_EXTPOWERFAILURE\_FALSE, soit Y\_EXTPOWERFAILURE\_TRUE

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

---

## **vsource.get\_failure()**

Indique si le module est en condition d'erreur.

**def get\_failure( )**

Il est possible de savoir de quelle erreur il s'agit en testant get\_overheat, get\_overcurrent etc... Lorsqu'une condition d'erreur est rencontrée, la tension de sortie est mise à zéro et ne peut pas être changée tant la fonction reset() n'aura pas été appelée.

**Retourne :**

soit Y\_FAILURE\_FALSE, soit Y\_FAILURE\_TRUE

---

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

---

### **`vsource.get_vsourceDescriptor()`**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

### **`vsource.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`.

---

### **`vsource.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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

---

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **vsourc.get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

**def get\_overCurrent( )**

**Retourne :**

soit Y\_OVERCURRENT\_FALSE, soit Y\_OVERCURRENT\_TRUE

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

---

### **vsourc.get\_overHeat()**

Rend TRUE si le module est en surchauffe.

**def get\_overHeat( )**

**Retourne :**

soit Y\_OVERHEAT\_FALSE, soit Y\_OVERHEAT\_TRUE

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

---

### **vsourc.get\_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

**def get\_overLoad( )**

**Retourne :**

soit Y\_OVERLOAD\_FALSE, soit Y\_OVERLOAD\_TRUE

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

---

### **vsourc.get\_regulationFailure()**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

**def get\_regulationFailure( )**

**Retourne :**

soit Y\_REGULATIONFAILURE\_FALSE, soit Y\_REGULATIONFAILURE\_TRUE

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

---

### **vsourc.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`.

---

### **`vsource.getUserData()`**

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

**`def getUserData( )`**

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.

---

### **`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`.

---

### **`vsource.isOnline()`**

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

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**`callback`** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**`context`** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**  
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

### **`vsource.load()`**

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

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

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **vsources.nextVSource()**

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.

---

## **vsources.pulse()**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

**def 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.

---

## **vsources.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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## **vsources.reset()**

Réinitialise la sortie du module.

```
def reset( )
```

Cette fonction doit être appelée après une condition d'erreur. Après toute condition d'erreur, le voltage de sortie est mis à zéro et ne peut pas être changé tant que cette fonction n'aura pas été appelée.

### **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.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'opération se déroule sans erreur.

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

---

## **vsources.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

---

## **vsource.set\_voltage()**

Règle la tension de sortie du module (en milliVolts).

```
def set_voltage( newval)
```

**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()**

Déclenche une variation constante de la sortie vers une valeur donnée.

```
def 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.

## **3.22. Interface de la fonction Wireless**

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
from yocto_wireless import *
```

### **Fonction globales**

#### **yFindWireless(func)**

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

#### **yFirstWireless()**

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

### **Méthodes des objets YWireless**

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

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

#### **wireless→describe()**

Retourne un court texte décrivant la fonction.

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

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

#### **wireless→get\_channel()**

Retourne le numéro du canal 802.

#### **wireless→get\_errorMessage()**

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

#### **wireless→get\_errorType()**

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

**wireless→get\_functionDescriptor()**

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

**wireless→get\_hardwareId()**

Retourne l'identifiant unique de la fonction.

**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\_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 la fonction est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant la fonction 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 la fonction, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes de la fonction, 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`.

## **YWireless.FindWireless()**

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()**

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()**

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()**

Retourne un court texte décrivant la fonction.

def **describe**( )

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

**Retourne :**  
une chaîne de caractères décrivant la fonction

---

### **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()**

Retourne le numéro du canal 802.

**def get\_channel( )**

11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

**Retourne :**  
un entier représentant le numéro du canal 802

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

---

### **wireless.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

---

### **wireless.get\_errorType()**

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

**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 fonction

---

### **wireless.get\_wirelessDescriptor()**

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`

---

Retourne l'identifiant unique de la fonction.

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

**Retourne :**

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

**wireless.get\_linkQuality()**

Retourne la qualité de la connection, exprimée en pourcents.

**def `get_linkQuality()`**

**Retourne :**

un entier représentant la qualité de la connection, exprimée en pourcents

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

---

**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\_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`

---

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui



n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **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()**

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()**

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.

---

## **wireless.isOnline()**

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

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## **wireless.joinNetwork()**

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()**

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.

---

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

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

## wireless.nextWireless()

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()

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'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

## wireless.set\_logicalName()

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'opération se déroule sans erreur.

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

---

### **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

## 4. Index

A	
adhocNetwork	182
API	9
C	
calibrate	98
calibrateFromPoints	27
callbackLogin	119
CheckLogicalName	10
D	
describe	18
DisableExceptions	10
E	
EnableExceptions	10
EnableUSBHost	11
F	
FindAnButton	17
FindCarbonDioxide	27
FindColorLed	35
FindCurrent	43
FindDataLogger	52
FindDualPower	67
FindHubPort	74
FindHumidity	82
FindLed	90
FindLightSensor	98
FindModule	107
FindNetwork	118
FindPressure	131
FindRelay	139
FindServo	147
FindTemperature	155
FindVoltage	164
FindVSource	173
FindWireless	181
FirstAnButton	17
FirstCarbonDioxide	27
FirstColorLed	36
FirstCurrent	44
FirstDataLogger	53
FirstDualPower	68
FirstHubPort	75
FirstHumidity	82
FirstLed	90

FirstLightSensor	98
FirstModule	107
FirstNetwork	119
FirstPressure	131
FirstRelay	140
FirstServo	147
FirstTemperature	155
FirstVoltage	165
FirstVSource	173
FirstWireless	182
forgetAllDataStreams	53
FreeAPI	11
functionCount	108
functionId	108
functionName	108
functionValue	108
G	
get_adminPassword	119
get_advertisedValue	18
get_analogCalibration	18
get_autoStart	53
get_averageValue	61
get_baudRate	75
get_beacon	109
get_blinking	91
get_calibratedValue	18
get_calibrationMax	18
get_calibrationMin	19
get_callbackCredentials	120
get_callbackMaxDelay	120
get_callbackMinDelay	120
get_callbackUrl	120
get_channel	183
get_columnCount	63
get_columnNames	63
get_currentRawValue	28
get_currentRunIndex	54
get_currentValue	28
get_data	64
get_dataRows	64
get_dataRun	54
get_dataSamplesInterval	64
get_dataStreams	54
get_duration	61
get_enabled	75
get_errorMessage	19
get_errorType	19
get_extPowerFailure	174
get_extVoltage	69
get_failure	174
get_firmwareRelease	109
get_functionDescriptor	19
get_hardwareId	20
get_highestValue	29
get_hslColor	37
get_icon2d	110
get_ipAddress	121
get_isPressed	20
get_lastTimePressed	20
get_lastTimeReleased	20
get_linkQuality	184
get_logicalName	20
get_lowestValue	30
get_luminosity	92

get_macAddress	122
get_maxValue	61
get_measureNames	56
get_minValue	61
get_module	21
get_module_async	21
get_neutral	149
get_oldestRunIndex	56
get_output	142
get_overCurrent	176
get_overHeat	176
get_overLoad	176
get_persistentSettings	110
get_portState	77
get_position	150
get_power	93
get_powerControl	70
get_powerState	70
get_primaryDNS	122
get_productId	111
get_productName	111
get_productRelease	111
get_pulseTimer	142
get_range	150
get_rawValue	21
get_readiness	123
get_rebootCountdown	111
get_recording	56
get_regulationFailure	176
get_resolution	30
get_rgbColor	38
get_rgbColorAtPowerOn	38
get_router	123
get_rowCount	65
get_runIndex	65
get_secondaryDNS	123
get_security	185
get_sensitivity	21
get_sensorType	159
get_serialNumber	111
get_ssid	185
get_startTime	65
get_startTimeUTC	62
get_state	142
get_subnetMask	123
get_timeUTC	57
get_unit	30
get_upTime	111
get_usbBandwidth	112
get_usbCurrent	112
get_userData	21
get_userPassword	124
get_valueCount	62
get_valueInterval	62
get_voltage	177
GetAPIVersion	11
GetTickCount	11
H	
HandleEvents	11
hslMove	38
I	
InitAPI	12
isOnline	22
isOnline_async	22

J	
joinNetwork	186
L	
load	22
load_async	22
M	
move	151
N	
nextAnButton	23
nextCarbonDioxide	32
nextColorLed	40
nextCurrent	49
nextDataLogger	58
nextDualPower	72
nextHubPort	79
nextHumidity	87
nextLed	94
nextLightSensor	104
nextModule	113
nextNetwork	125
nextPressure	136
nextRelay	144
nextServo	152
nextTemperature	161
nextVoltage	170
nextVSource	178
nextWireless	187
P	
pulse	144
R	
reboot	114
RegisterDeviceArrivalCallback	12
RegisterDeviceRemovalCallback	12
RegisterHub	13
RegisterLogFunction	13
registerValueCallback	23
reset	179
revertFromFlash	114
rgbMove	40
S	
saveToFlash	114
set_adminPassword	126
set_analogCalibration	23
set_autoStart	59
set_beacon	114
set_blinking	95
set_calibrationMax	24
set_calibrationMin	24
set_callbackCredentials	126
set_callbackMaxDelay	126
set_callbackMinDelay	127
set_callbackUrl	127
set_enabled	79
set_highestValue	33
set_hslColor	41
set_logicalName	24
set_lowestValue	33
set_luminosity	95
set_neutral	152
set_output	145
set_position	153
set_power	96
set_powerControl	72
set_primaryDNS	127



set_range	153
set_recording	59
set_rgbColor	41
set_rgbColorAtPowerOn	41
set_secondaryDNS	128
set_sensitivity	24
set_sensorType	162
set_state	145
set_timeUTC	59
set_usbBandwidth	115
set_userData	25
set_userPassword	128
set_valueInterval	62
set_voltage	180
SetDelegate	13
SetTimeout	14
Sleep	14
T	
triggerFirmwareUpdate	116
U	
UnregisterHub	14
UpdateDeviceList	14
UpdateDeviceList_async	15
useDHCP	128
useStaticIP	129
V	
voltageMove	180
Y	
YAnButton	15
YCarbonDioxide	25
YColorLed	34
YCurrent	42
YDataLogger	50
YDataRun	60
YDataStream	63
YDualPower	66
YHubPort	73
YHumidity	80
YLed	89
YLightSensor	96
YModule	105
YNetwork	116
YPressure	129
YRelay	138
YServo	145
YTemperature	153
YVoltage	163
YVSource	171
YWireless	180