



## Référence de l'API C++



# Table des matières

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Utilisation du Yocto-Demo en C++ .....</b>	<b>3</b>
2.1. Contrôle de la fonction Led .....	3
2.2. Contrôle de la partie module .....	5
2.3. Gestion des erreurs .....	7
2.4. Intégration de la librairie Yoctopuce en C++ .....	8
Blueprint .....	12
<b>3. Reference .....</b>	<b>12</b>
3.1. Fonctions générales .....	13
3.2. Interface de la fonction Accelerometer .....	33
3.3. Interface de la fonction AnButton .....	75
3.4. Interface de la fonction CarbonDioxide .....	113
3.5. Interface de la fonction ColorLed .....	152
3.6. Interface de la fonction Compass .....	181
3.7. Interface de la fonction Current .....	221
3.8. Interface de la fonction DataLogger .....	260
3.9. Séquence de données mise en forme .....	291
3.10. Séquence de données enregistrées .....	293
3.11. Séquence de données enregistrées brute .....	305
3.12. Interface de la fonction DigitalIO .....	320
3.13. Interface de la fonction Display .....	364
3.14. Interface des objets DisplayLayer .....	411
3.15. Interface de contrôle de l'alimentation .....	443
3.16. Interface de la fonction Files .....	468
3.17. Interface de la fonction GenericSensor .....	496
3.18. Interface de la fonction Gyro .....	542
3.19. Interface d'un port de Yocto-hub .....	593
3.20. Interface de la fonction Humidity .....	618
3.21. Interface de la fonction Led .....	657
3.22. Interface de la fonction LightSensor .....	684
3.23. Interface de la fonction Magnetometer .....	724
3.24. Valeur mesurée .....	766
3.25. Interface de contrôle du module .....	772

3.26. Interface de la fonction Network .....	815
3.27. contrôle d'OS .....	872
3.28. Interface de la fonction Power .....	895
3.29. Interface de la fonction Pressure .....	938
3.30. Interface de la fonction Pwm .....	977
3.31. Interface de la fonction PwmPowerSource .....	1015
3.32. Interface du quaternion .....	1038
3.33. Interface de la fonction Horloge Temps Real .....	1077
3.34. Configuration du référentiel .....	1104
3.35. Interface de la fonction Relay .....	1140
3.36. Interface des fonctions de type senseur .....	1176
3.37. Interface de la fonction Servo .....	1215
3.38. Interface de la fonction Temperature .....	1250
3.39. Interface de la fonction Tilt .....	1291
3.40. Interface de la fonction Voc .....	1330
3.41. Interface de la fonction Voltage .....	1369
3.42. Interface de la fonction Source de tension .....	1408
3.43. Interface de la fonction WakeUpMonitor .....	1440
3.44. Interface de la fonction WakeUpSchedule .....	1475
3.45. Interface de la fonction Watchdog .....	1512
3.46. Interface de la fonction Wireless .....	1557
<b>Index .....</b>	<b>1587</b>

# 1. Introduction

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

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

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



## 2. Utilisation du Yocto-Demo en C++

Le C++ n'est pas le langage le plus simple à maîtriser. Pourtant, si on prend soin à se limiter aux fonctionnalités essentielles, c'est un langage tout à fait utilisable pour des petits programmes vite faits, et qui a l'avantage d'être très portable d'un système d'exploitation à l'autre. Sous Windows, tous les exemples et les modèles de projet sont testés avec Microsoft Visual Studio 2010 Express, disponible gratuitement sur le site de Microsoft<sup>1</sup>. Sous Mac OS X, tous les exemples et les modèles de projet sont testés avec XCode 4, disponible sur l'App Store. Par ailleurs, aussi bien sous Mac OS X que sous Linux, vous pouvez compiler les exemples en ligne de commande avec GCC en utilisant le `GNUmakefile` fourni. De même, sous Windows, un `Makefile` pour permet de compiler les exemples en ligne de commande, et en pleine connaissance des arguments de compilation et link.

Les librairies Yoctopuce<sup>2</sup> pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis le C++. La librairie vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

Vous allez rapidement vous rendre compte que l'API C++ définit beaucoup de fonctions qui retournent des objets. Vous ne devez jamais désallouer ces objets vous-même. Ils seront désalloués automatiquement par l'API à la fin de l'application.

Afin de garder les choses simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soi que le fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique. Vous trouverez dans la dernière section de ce chapitre toutes les informations nécessaires à la création d'un projet à neuf lié avec les librairies Yoctopuce.

### 2.1. 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 C++ qui utilise la fonction Led.

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

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

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

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

```
// On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg);
led = yFindLed("YCTOPOC1-123456.led");

// Pour gérer le hot-plug, on vérifie que le module est là
if(led->isOnline())
{
    // Utiliser led->set_power(), ...
}
```

Voyons maintenant en détail ce que font ces quelques lignes.

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

Ces deux fichiers inclus permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api.h` doit toujours être utilisé, `yocto_led.h` est nécessaire pour gérer les modules contenant une led, comme le Yocto-Demo.

### yRegisterHub

La fonction `yRegisterHub` 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 le paramètre `errmsg` un explication du problème.

### yFindLed

La fonction `yFindLed`, 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):

```
YLed *led = yFindLed("YCTOPOC1-123456.led");
YLed *led = yFindLed("YCTOPOC1-123456.MaFonction");
YLed *led = yFindLed("MonModule.led");
YLed *led = yFindLed("MonModule.MaFonction");
YLed *led = yFindLed("MaFonction");
```

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

### isOnline

La méthode `isOnline()` de l'objet renvoyé par `yFindLed` 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 `yFindLed` permet d'allumer et d'éteindre la led. L'argument est `Y_POWER_ON` ou `Y_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 environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce. Si vous préférez travailler avec votre éditeur de texte préféré, ouvrez le fichier `main.cpp`, vous taperez simplement `make` dans le répertoire de l'exemple pour le compiler.

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.

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

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

using namespace std;

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

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

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

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

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

    return 0;
}
```

## 2.2. 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.

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

#include "yocto_api.h"

using namespace std;

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

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

```

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

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

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

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

```

Chaque propriété `xxx` du module peut être lue grâce à une méthode du type `get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `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 `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'elles 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 `saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```

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

#include "yocto_api.h"

using namespace std;

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

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

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

```

```

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

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

if (module->isOnline()) {
    if (argc >= 3) {
        string newname = argv[2];
        if (!yCheckLogicalName(newname)) {
            cerr << "Invalid name (" << newname << ")" << endl;
            usage(argv[0]);
        }
        module->set_logicalName(newname);
        module->saveToFlash();
    }
    cout << "Current name: " << module->get_logicalName() << endl;
} else {
    cout << argv[1] << " not connected (check identification and USB cable)"
    << endl;
}
return 0;
}

```

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 alentour de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `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 `yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `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 module connectés

```

#include <iostream>
#include "yocto_api.h"
using namespace std;

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

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

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

```

## 2.3. 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 renvoyée 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 renvoyée. 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 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.

## 2.4. Intégration de la librairie Yoctopuce en C++

Selon vos besoins et vos préférences, vous pouvez être amené à intégrer de différentes manières la librairie à vos projets. Cette section explique comment implémenter les différentes options.

### Intégration au format source

L'intégration de toutes les sources de la librairie dans vos projets a plusieurs avantages:

- Elle garantit le respect des conventions de compilation de votre projet (32/64 bits, inclusion des symboles de debug, caractères unicode ou ASCII, etc.);
- Elle facilite le débogage si vous cherchez la cause d'un problème lié à la librairie Yoctopuce;
- Elle réduit les dépendances sur des composants tiers, par exemple pour parer au cas où vous pourriez être amené à recompiler ce projet pour une architecture différente dans de nombreuses années.

- Elle ne requiert pas l'installation d'une librairie dynamique spécifique à Yoctopuce sur le système final, tout est dans l'exécutable.

Pour intégrer le code source, le plus simple est d'inclure simplement le répertoire Sources de la librairie Yoctopuce à votre **IncludePath**, et d'ajouter tous les fichiers de ce répertoire (y compris le sous-répertoire `yapi`) à votre projet.

Pour que votre projet se construise ensuite correctement, il faudra linker avec votre projet les librairies systèmes requises, à savoir:

- Pour Windows: les librairies sont mises automatiquement
- Pour Mac OS X: **IOKit.framework** et **CoreFoundation.framework**
- Pour Linux: **libm**, **libpthread**, **libusb1.0** et **libstdc++**

## Intégration en librairie statique

L'intégration de la librairie Yoctopuce sous forme de librairie statique est une manière plus simple de construire un petit exécutable utilisant des modules Yoctopuce. Elle permet une compilation rapide du programme en une seule commande. Elle ne requiert pas non plus l'installation d'une librairie dynamique spécifique à Yoctopuce sur le système final, tout est dans l'exécutable.

Pour intégrer la librairie statique Yoctopuce à votre projet, vous devez inclure le répertoire Sources de la librairie Yoctopuce à votre **IncludePath**, et ajouter le sous-répertoire de Binaries/... correspondant à votre système d'exploitation à votre **LibPath**.

Ensuite, pour que votre projet se construise ensuite correctement, il faudra linker avec votre projet la librairie Yoctopuce et les librairies systèmes requises:

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

Attention, sous Linux, si vous voulez compiler en ligne de commande avec GCC, il est en général souhaitable de linker les librairies systèmes en dynamique et non en statique. Pour mélanger sur la même ligne de commande des librairies statiques et dynamiques, il faut passer les arguments suivants:

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

## Intégration en librairie dynamique

L'intégration de la librairie Yoctopuce sous forme de librairie dynamique permet de produire un exécutable plus petit que les deux méthodes précédentes, et de mettre éventuellement à jour cette librairie si un correctif s'avérait nécessaire sans devoir recompiler le code source de l'application. Par contre, c'est un mode d'intégration qui exigera systématiquement de copier la librairie dynamique sur la machine cible ou l'application devra être lancée (**yocto.dll** sous Windows, **libyocto.so.1.0.1** sous Mac OS X et Linux).

Pour intégrer la librairie dynamique Yoctopuce à votre projet, vous devez inclure le répertoire Sources de la librairie Yoctopuce à votre **IncludePath**, et ajouter le sous-répertoire de Binaries/... correspondant à votre système d'exploitation à votre **LibPath**.

Ensuite, pour que votre projet se construise ensuite correctement, il faudra linker avec votre projet la librairie dynamique Yoctopuce et les librairies systèmes requises:

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

Avec GCC, la ligne de commande de compilation est simplement:

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



### **3. Reference**

## 3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### `yCheckLogicalName(name)`

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

#### `yDisableExceptions()`

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

#### `yEnableExceptions()`

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

#### `yEnableUSBHost(osContext)`

Cette fonction est utilisée uniquement sous Android.

#### `yFreeAPI()`

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

#### `yGetAPIVersion()`

Retourne la version de la librairie Yoctopuce utilisée.

#### `yGetTickCount()`

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

#### `yHandleEvents(errmsg)`

Maintient la communication de la librairie avec les modules Yoctopuce.

#### `yInitAPI(mode, errmsg)`

Initialise la librairie de programmation de Yoctopuce explicitement.

#### `yPreregisterHub(url, errmsg)`

Alternative plus tolérante à `RegisterHub()`.

#### `yRegisterDeviceArrivalCallback(arrivalCallback)`

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

#### `yRegisterDeviceRemovalCallback(removalCallback)`

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

#### `yRegisterHub(url, errmsg)`

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

#### `yRegisterHubDiscoveryCallback(hubDiscoveryCallback)`

### 3. Reference

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

#### yRegisterLogFunction(logfun)

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

#### ySelectArchitecture(arch)

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

#### ySetDelegate(object)

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

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

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

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

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

#### yTriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

#### yUnregisterHub(url)

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

#### yUpdateDeviceList(errmsg)

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

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

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

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

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

```
bool yCheckLogicalName( const string& name)
```

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

**Paramètres :**

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

**Retourne :**

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

## **YAPI.DisableExceptions()**

**YAPI**

### **yDisableExceptions()yDisableExceptions( )**

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

```
void yDisableExceptions( )
```

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

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

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

```
void yEnableExceptions( )
```

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

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

---

**YAPI**

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

```
void yFreeAPI( )
```

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

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

Retourne la version de la librairie Yoctopuce utilisée.

```
string yGetAPIVersion( )
```

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

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

**Retourne :**

une chaîne de caractères décrivant la version de la librairie.

## YAPI.GetTickCount()

YAPI

### yGetTickCount()yGetTickCount( )

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

u64 yGetTickCount( )

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()****YAPI****yHandleEvents()yHandleEvents( )**

Maintient la communication de la librairie avec les modules Yoctopuce.

**YRETCODE yHandleEvents( string& errmsg)**

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

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

**Paramètres :**

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

**Retourne :**

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

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

YAPI

Initialise la librairie de programmation de Yoctopuce explicitement.

**YRETCODE yInitAPI( int mode, string& errmsg)**

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

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

### Paramètres :

**mode** un entier spécifiant le type de détection automatique de modules à utiliser. Les valeurs possibles sont `Y_DETECT_NONE`, `Y_DETECT_USB`, `Y_DETECT_NET` et `Y_DETECT_ALL`.

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

### Retourne :

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

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

Alternative plus tolérante à RegisterHub().

**YRETCODE yPreregisterHub( const string& url, string& errmsg)**

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

**Paramètres :**

**url** une chaîne de caractères contenant "usb", "callback", ou l'URL racine du VirtualHub à utiliser.

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

**Retourne :**

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

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

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

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

```
void yRegisterDeviceArrivalCallback( yDeviceUpdateCallback arrivalCallback)
```

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

**Paramètres :**

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

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

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

```
void yRegisterDeviceRemovalCallback( yDeviceUpdateCallback removalCallback)
```

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

**Paramètres :**

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

**YAPI.RegisterHub()**

YAPI

**yRegisterHub()yRegisterHub( )**

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

**YRETCODE yRegisterHub( const string& url, string& errmsg)**

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

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

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

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

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

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

http://nom:mot\_de\_passe@adresse:port

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

**Paramètres :**

**url** une chaîne de caractères contenant "**usb**", "**callback**", ou l'URL racine du VirtualHub à utiliser.

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

**Retourne :**

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

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

**YAPI.RegisterHubDiscoveryCallback()**  
**yRegisterHubDiscoveryCallback()**  
**yRegisterHubDiscoveryCallback( )****YAPI**

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

```
void yRegisterHubDiscoveryCallback( YHubDiscoveryCallback hubDiscoveryCallback)
```

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

**Paramètres :**

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

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

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

```
void yRegisterLogFunction( yLogFunction logfun)
```

Utile pour débugger le fonctionnement de l'API.

**Paramètres :**

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

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

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

**YRETCODE ySleep( unsigned ms\_duration, string& errmsg)**

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 où la communication avec un module Yoctopuce ne se passerait pas comme attendu.

**Paramètres :**

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

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

**Retourne :**

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

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

Relance une détection des hubs réseau.

**YRETCODE yTriggerHubDiscovery( string& errmsg)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

```
void yUnregisterHub( const string& url)
```

**Paramètres :**

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

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

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

**YRETCODE yUpdateDeviceList( string& errmsg)**

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

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

**Paramètres :**

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

**Retourne :**

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

## 3.2. Interface de la fonction Accelerometer

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### yFindAccelerometer(func)

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

#### yFirstAccelerometer()

Commence l'énumération des accéléromètres accessibles par la librairie.

### Méthodes des objets YAccelerometer

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

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

#### accelerometer→describe()

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

#### accelerometer→get\_advertisedValue()

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

#### accelerometer→get\_currentRawValue()

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

#### accelerometer→get\_currentValue()

Retourne la valeur actuelle de l'accélération.

#### accelerometer→get\_errorMessage()

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

#### accelerometer→get\_errorType()

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

#### accelerometer→get\_friendlyName()

Retourne un identifiant global de l'accéléromètre au format NOM\_MODULE . NOM\_FONCTION.

#### accelerometer→get\_functionDescriptor()

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

#### accelerometer→get\_functionId()

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

#### accelerometer→get\_hardwareId()

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

### 3. Reference

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

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

#### **accelerometer→get\_logFrequency()**

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

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

Retourne le nom logique de l'accéléromètre.

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

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

#### **accelerometer→get\_module()**

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

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

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

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

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

#### **accelerometer→get\_reportFrequency()**

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

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

Retourne la résolution des valeurs mesurées.

#### **accelerometer→get\_unit()**

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

#### **accelerometer→get\_userData()**

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

#### **accelerometer→get\_xValue()**

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

#### **accelerometer→get\_yValue()**

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

#### **accelerometer→get\_zValue()**

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

#### **accelerometer→isOnline()**

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

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

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

#### **accelerometer→load(msValidity)**

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

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

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

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

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

#### **accelerometer→nextAccelerometer()**

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

#### **accelerometer→registerTimedReportCallback(callback)**

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

**accelerometer→registerValueCallback(callback)**

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

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

Modifie la mémoire de valeur maximale observée.

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

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

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

Modifie le nom logique de l'accéléromètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**accelerometer→set(userData)**

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

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

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

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

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

**YAccelerometer\* yFindAccelerometer( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

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

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

**Paramètres :**

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

**Retourne :**

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

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

Commence l'énumération des accéléromètres accessibles par la librairie.

**YAccelerometer\* yFirstAccelerometer( )**

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

**Retourne :**

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

**accelerometer→calibrateFromPoints()****YAccelerometer****accelerometer→calibrateFromPoints( )**

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

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

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

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

**Retourne :**

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

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

**accelerometer→describe()accelerometer→  
describe( )**

**YAccelerometer**

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

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant l'accéléromètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**accelerometer→get\_advertisedValue()**

**YAccelerometer**

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

---

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

**string get\_advertisedValue( )**

**Retourne :**

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

---

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

---

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

```
double get_currentRawValue( )
```

**Retourne :**

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

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

**accelerometer→get\_currentValue()**

**YAccelerometer**

**accelerometer→currentValue()accelerometer→**

**get\_currentValue( )**

---

Retourne la valeur actuelle de l'accélération.

**double get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'accélération

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

**accelerometer→getErrorMessage()**

**YAccelerometer**

**accelerometer→errorMessage()accelerometer→  
getErrorMessage( )**

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

**string getErrorMessage( )**

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

**Retourne :**

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

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

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

**YRETCODE get\_errorType( )**

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

**Retourne :**

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

**accelerometer→get\_friendlyName()**

**YAccelerometer**

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

Retourne un identifiant global de l'accéléromètre au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

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

**Retourne :**

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

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

---

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

**YFUN\_DESCR get\_functionDescriptor( )**

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

**Retourne :**

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

**accelerometer→get\_functionId()**

**YAccelerometer**

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

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

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

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

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

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

**string get\_hardwareId( )**

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

**Retourne :**

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

**accelerometer→get\_highestValue()**

**YAccelerometer**

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

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

**double get\_highestValue( )**

**Retourne :**

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

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

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

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

string **get\_logFrequency( )**

**Retourne :**

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

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

---

<b>accelerometer→get_logicalName()</b>	<b>YAccelerometer</b>
<b>accelerometer→logicalName()accelerometer→get_logicalName( )</b>	

---

Retourne le nom logique de l'accéléromètre.

```
string get_logicalName( )
```

**Retourne :**

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

**accelerometer→get\_lowestValue()**

**YAccelerometer**

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

---

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

```
double get_lowestValue( )
```

**Retourne :**

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

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

**accelerometer→get\_module()**

**YAccelerometer**

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

---

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

**YModule \* get\_module( )**

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

**Retourne :**

une instance de **YModule**

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

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

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

**accelerometer→get\_reportFrequency()**

**YAccelerometer**

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

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

**string get\_reportFrequency( )**

**Retourne :**

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

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

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

**YAccelerometer**

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

**accelerometer→get\_unit()**

**YAccelerometer**

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

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

**string get\_unit( )**

**Retourne :**

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

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

**accelerometer→get(userData)**

**YAccelerometer**

**accelerometer→userData()accelerometer→**

**get(userData()**

---

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

**void \* get(userData( )**

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

**accelerometer→get\_xValue()**

**YAccelerometer**

**accelerometer→xValue()accelerometer→**

**get\_xValue( )**

---

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

**double get\_xValue( )**

**Retourne :**

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

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

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

---

**YAccelerometer**

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

**double get\_yValue( )**

**Retourne :**

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

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

**accelerometer→get\_zValue()**

**YAccelerometer**

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

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

**double get\_zValue( )**

**Retourne :**

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

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

accelerometer → **isOnline()** accelerometer →  
**isOnline()**

YAccelerometer

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

**bool isOnline( )**

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

**Retourne :**

true si l'accéléromètre est joignable, false sinon

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

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

**YRETCODE load( int msValidity)**

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

**Paramètres :**

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

**Retourne :**

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

**accelerometer→loadCalibrationPoints()****YAccelerometer****accelerometer→loadCalibrationPoints( )**

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

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

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

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

**Retourne :**

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

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

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

**YAccelerometer**

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

`YAccelerometer * nextAccelerometer( )`

**Retourne :**

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

**accelerometer→registerTimedReportCallback()**

**YAccelerometer**

**accelerometer→**

**registerTimedReportCallback( )**

---

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

**int registerTimedReportCallback( YAccelerometerTimedReportCallback **callback**)**

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

**Paramètres :**

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

**accelerometer→registerValueCallback()**  
**accelerometer→registerValueCallback( )**

**YAccelerometer**

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

```
int registerValueCallback( YAccelerometerValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

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

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

---

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

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

**Retourne :**

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

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

**accelerometer→set\_logFrequency()**

**YAccelerometer**

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

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

**int set\_logFrequency( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie le nom logique de l'accéléromètre.

```
int set_logicalName( const string& newval)
```

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

**Paramètres :**

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

**Retourne :**

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

---

<b>accelerometer→set_lowestValue()</b>	<b>YAccelerometer</b>
<b>accelerometer→setLowestValue()accelerometer→</b> <b>set_lowestValue( )</b>	

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**accelerometer→set\_resolution()**

**YAccelerometer**

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

---

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**accelerometer→set(userData)**

**YAccelerometer**

**accelerometer→setUserData()accelerometer→**

**set(userData()**

---

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

**void set(userData( void\* data)**

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

**Paramètres :**

**data** objet quelconque à mémoriser

### 3.3. Interface de la fonction AnButton

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

#### Fonction globales

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

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

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

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

#### Méthodes des objets YAnButton

##### **anbutton→describe()**

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

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

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

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

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

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

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

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

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

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

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

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

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

##### **anbutton→get\_errorType()**

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

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

Retourne un identifiant global de l'entrée analogique au format NOM\_MODULE.NOM\_FONCTION.

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

### 3. Reference

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>anbutton→get_functionId()</b> Retourne l'identifiant matériel de l'entrée analogique, sans référence au module.
<b>anbutton→get_hardwareId()</b> Retourne l'identifiant matériel unique de l'entrée analogique au format SERIAL . FUNCTIONID.
<b>anbutton→get_isPressed()</b> Retourne vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon.
<b>anbutton→get_lastTimePressed()</b> 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).
<b>anbutton→get_lastTimeReleased()</b> 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).
<b>anbutton→get_logicalName()</b> Retourne le nom logique de l'entrée analogique.
<b>anbutton→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>anbutton→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>anbutton→get_pulseCounter()</b> Retourne la valeur du compteur d'impulsions.
<b>anbutton→get_pulseTimer()</b> Retourne le timer du compteur d'impulsions (ms)
<b>anbutton→get_rawValue()</b> Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).
<b>anbutton→get_sensitivity()</b> Retourne la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.
<b>anbutton→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>anbutton→isOnline()</b> Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.
<b>anbutton→isOnline_async(callback, context)</b> Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.
<b>anbutton→load(msValidity)</b> Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.
<b>anbutton→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.
<b>anbutton→nextAnButton()</b> Continue l'énumération des entrées analogiques commencée à l'aide de yFirstAnButton( ).
<b>anbutton→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>anbutton→resetCounter()</b> réinitialise le compteur d'impulsions et son timer
<b>anbutton→set_analogCalibration(newval)</b> Enclenche ou déclenche le procédure de calibration.
<b>anbutton→set_calibrationMax(newval)</b>

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

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

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

**anbutton→set\_logicalName(newval)**

Modifie le nom logique de l'entrée analogique.

**anbutton→set\_sensitivity(newval)**

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

**anbutton→set\_userData(data)**

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

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

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

**YAnButton.FindAnButton()****YAnButton****yFindAnButton()yFindAnButton( )**

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

**YAnButton\* yFindAnButton( const string& 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()****yFirstAnButton()yFirstAnButton( )****YAnButton**

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

**YAnButton\* yFirstAnButton( )**

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()****YAnButton**

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

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant l'entrée analogique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

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

**YAnButton**

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

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

---

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

**Y\_ANALOGCALIBRATION\_enum 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()**  
**anbutton→calibratedValue()** anbutton→  
**get\_calibratedValue( )**

**YAnButton**

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

**int get\_calibratedValue( )**

**Retourne :**

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

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

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

---

**YAnButton**

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

**int get\_calibrationMax( )**

**Retourne :**

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

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

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

**YAnButton**

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

**int 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()**  
**anbutton→errorMessage()****anbutton→get\_errorMessage( )**

**YAnButton**

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

**string get\_errorMessage( )**

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

**Retourne :**

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

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

**YAnButton**

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

**YRETCODE get\_errorType( )**

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

**Retourne :**

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

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

---

Retourne un identifiant global de l'entrée analogique au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

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

**Retourne :**

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

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

**YAnButton**

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

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

---

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

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

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

---

<b>anbutton→get_hardwareId()</b>	<b>YAnButton</b>
<b>anbutton→hardwareId()anbutton→ get_hardwareId( )</b>	

---

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

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'entrée analogique (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'entrée analogique (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**anbutton→get\_isPressed()**

**YAnButton**

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

---

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

**Y\_ISPRESSED\_enum 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()**  
**anbutton→lastTimePressed()anbutton→**  
**get\_lastTimePressed( )**

**YAnButton**

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).

**s64 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.

---

<b>anbutton→get_lastTimeReleased()</b>	<b>YAnButton</b>
<b>anbutton→lastTimeReleased()</b>	<b>anbutton→</b>
<b>get_lastTimeReleased( )</b>	

---

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).

**s64 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()**  
**anbutton→logicalName()****anbutton→get\_logicalName( )**

**YAnButton**

Retourne le nom logique de l'entrée analogique.

**string get\_logicalName( )**

**Retourne :**

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

**anbutton→get\_module()**

**YAnButton**

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

---

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

**YModule \* get\_module( )**

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

**Retourne :**

une instance de YModule

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

**YAnButton**

Retourne la valeur du compteur d'impulsions.

s64 **get\_pulseCounter( )**

**Retourne :**

un entier représentant la valeur du compteur d'impulsions

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

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

---

**YAnButton**

Retourne le timer du compteur d'impulsions (ms)

s64 **get\_pulseTimer( )**

**Retourne :**

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

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

---

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

---

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).**int get\_rawValue( )****Retourne :**

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

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

**anbutton→get\_sensitivity()**

**YAnButton**

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

---

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

**int get\_sensitivity( )**

**Retourne :**

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

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

**anbutton→get(userData)****YAnButton****anbutton→userData()anbutton→get(userData())**

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

```
void * 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()**

**YAnButton**

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

**bool isOnline( )**

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

**Retourne :**

true si l'entrée analogique est joignable, false sinon

**anbutton→load()****YAnButton**

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

**YRETCODE load( int msValidity)**

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

**Paramètres :**

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

**Retourne :**

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

**anbutton→nextAnButton()**  
**anbutton→**  
**nextAnButton( )**

**YAnButton**

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

`YAnButton * nextAnButton()`

**Retourne :**

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

**anbutton→registerValueCallback()**  
**anbutton→registerValueCallback( )**

**YAnButton**

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

```
int registerValueCallback( YAnButtonValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

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

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

---

**YAnButton**

réinitialise le compteur d'impulsions et son timer

**int resetCounter( )**

**Retourne :**

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

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

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

**YAnButton**

Enclenche ou déclenche le procédure de calibration.

```
int set_analogCalibration( Y_ANALOGCALIBRATION_enum newval)
```

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

**Paramètres :**

**newval** soit Y\_ANALOGCALIBRATION\_OFF, soit Y\_ANALOGCALIBRATION\_ON

**Retourne :**

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

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

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

**YAnButton**

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

**int set\_calibrationMax( int newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

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

**int set\_calibrationMin( int newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

Modifie le nom logique de l'entrée analogique.

**int set\_logicalName( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**YAnButton**

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

**int set\_sensitivity( int newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**anbutton→set(userData)**  
**anbutton→setUserData()****anbutton→**  
**set(userData)**

**YAnButton**

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).

**void set(userData void\* 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 CarbonDioxide

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### **yFindCarbonDioxide(func)**

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

#### **yFirstCarbonDioxide()**

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

### Méthodes des objets YCarbonDioxide

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

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

#### **carbondioxide→describe()**

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

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

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

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

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

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

Retourne la valeur actuelle du taux de CO2.

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

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

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

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

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

Retourne un identifiant global du capteur de CO2 au format NOM\_MODULE . NOM\_FONCTION.

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

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

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

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

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

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

### 3. Reference

#### **carbon dioxide → get\_highestValue()**

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

#### **carbon dioxide → get\_logFrequency()**

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

#### **carbon dioxide → get\_logicalName()**

Retourne le nom logique du capteur de CO2.

#### **carbon dioxide → get\_lowestValue()**

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

#### **carbon dioxide → get\_module()**

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

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

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

#### **carbon dioxide → get\_recordedData(startTime, endTime)**

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

#### **carbon dioxide → get\_reportFrequency()**

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

#### **carbon dioxide → get\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **carbon dioxide → get\_unit()**

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

#### **carbon dioxide → get\_userData()**

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

#### **carbon dioxide → isOnline()**

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

#### **carbon dioxide → isOnline\_async(callback, context)**

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

#### **carbon dioxide → load(msValidity)**

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

#### **carbon dioxide → loadCalibrationPoints(rawValues, refValues)**

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

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

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

#### **carbon dioxide → nextCarbonDioxide()**

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

#### **carbon dioxide → registerTimedReportCallback(callback)**

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

#### **carbon dioxide → registerValueCallback(callback)**

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

#### **carbon dioxide → set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

#### **carbon dioxide → set\_logFrequency(newval)**

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

**carbon dioxide → set\_logicalName(newval)**

Modifie le nom logique du capteur de CO2.

**carbon dioxide → set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**carbon dioxide → set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**carbon dioxide → set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**carbon dioxide → 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).

**carbon dioxide → wait\_async(callback, context)**

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

## YCarbonDioxide.FindCarbonDioxide()

## YCarbonDioxide

### yFindCarbonDioxide()yFindCarbonDioxide( )

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

**YCarbonDioxide\* yFindCarbonDioxide( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

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

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide\* yFirstCarbonDioxide( )**

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

**Retourne :**

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

**carbon dioxide → calibrateFromPoints()****YCarbonDioxide****carbon dioxide → calibrateFromPoints( )**

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

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**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()carbon dioxide→  
describe( )**

**YCarbonDioxide**

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

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de CO2 (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**carbondioxide→get\_advertisedValue()**

**YCarbonDioxide**

**carbondioxide→advertisedValue()carbon dioxide→  
get\_advertisedValue( )**

---

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

**string 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.

---

<b>carbondioxide→get_currentRawValue()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→currentRawValue()carbon dioxide</b>	

---

→**get\_currentRawValue( )**

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

**double 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()**

**YCarbonDioxide**

**carbondioxide→currentValue()carbondioxide→**

**get\_currentValue( )**

---

Retourne la valeur actuelle du taux de CO2.

double **get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur actuelle du taux de CO2

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

**carbondioxide→getErrorMessage()****YCarbonDioxide****carbondioxide→errorMessage()carbondioxide→  
getErrorMessage( )**

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

```
string getErrorMessage( )
```

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

**Retourne :**

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

**carbon dioxide → get\_errorType()**

**YCarbonDioxide**

**carbon dioxide → errorType() carbon dioxide →**

**get\_errorType( )**

---

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

**YRETCode get\_errorType( )**

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

**Retourne :**

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

---

<b>carbondioxide→get_friendlyName()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→friendlyName()carbon dioxide→get_friendlyName( )</b>	

---

Retourne un identifiant global du capteur de CO2 au format NOM\_MODULE.NOM\_FONCTION.

```
string get_friendlyName( )
```

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

**Retourne :**

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

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

---

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

**YFUN\_DESCR 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

---

<b>carbondioxide→get_functionId()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→functionId()carbondioxide→</b>	
<b>get_functionId( )</b>	

---

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

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

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

**carbon dioxide → get.hardwareId()**

**YCarbonDioxide**

**carbon dioxide → hardwareId() carbon dioxide →**

**get.hardwareId( )**

---

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

**string get.hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de CO2 (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de CO2 (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**carbondioxide→get\_highestValue()**

**YCarbonDioxide**

**carbondioxide→highestValue()carbon dioxide→  
get\_highestValue()**

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

**double get\_highestValue( )**

**Retourne :**

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

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

**carbon dioxide → get\_logFrequency()**

**YCarbonDioxide**

**carbon dioxide → logFrequency() carbon dioxide →**

**get\_logFrequency( )**

---

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

**string get\_logFrequency( )**

**Retourne :**

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

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

---

<b>carbondioxide→get_logicalName()</b>	<b>YCarbonDioxide</b>
<b>carbondioxide→logicalName()carbon dioxide→get_logicalName( )</b>	

---

Retourne le nom logique du capteur de CO2.

```
string get_logicalName( )
```

**Retourne :**

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

**carbondioxide→get\_lowestValue()**

**YCarbonDioxide**

**carbondioxide→lowestValue()carbondioxide→**

**get\_lowestValue()**

---

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

```
double get_lowestValue( )
```

**Retourne :**

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

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

**carbondioxide→get\_module()****YCarbonDioxide****carbondioxide→module()carbon dioxide→  
get\_module()**

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

`YModule * get_module( )`

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

**Retourne :**

une instance de `YModule`

<b>carbon dioxide → get_recordedData()</b>	<b>YCarbonDioxide</b>
<b>carbon dioxide → recordedData() carbon dioxide →</b>	

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

#### **YDataSet get\_recordedData( s64 startTime, s64 endTime)**

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

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

#### **Paramètres :**

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

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

#### **Retourne :**

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

---

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

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

```
string get_reportFrequency( )
```

**Retourne :**

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

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

**carbondioxide→get\_resolution()**

**YCarbonDioxide**

**carbondioxide→resolution()carbon dioxide→  
get\_resolution( )**

---

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

**carbondioxide→get\_unit()****YCarbonDioxide****carbondioxide→unit()carbon dioxide→get\_unit()**

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

```
string get_unit( )
```

**Retourne :**

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

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

**carbondioxide→get(userData)**

**YCarbonDioxide**

**carbondioxide→userData()carbon dioxide→**

**get(userData)**

---

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

**void \* 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()** **carbondioxide→  
isOnline( )**

**YCarbonDioxide**

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

**bool isOnline( )**

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

**Retourne :**

true si le capteur de CO2 est joignable, false sinon

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

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

**YRETCODE load( int msValidity)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

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

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

**Retourne :**

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

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

**carbondioxide→nextCarbonDioxide()**  
**carbondioxide→nextCarbonDioxide( )**

---

**YCarbonDioxide**

Continue l'énumération des capteurs de CO<sub>2</sub> commencée à l'aide de `yFirstCarbonDioxide()`.

`YCarbonDioxide * nextCarbonDioxide( )`

**Retourne :**

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

**carbondioxide→registerTimedReportCallback()**

**YCarbonDioxide**

**carbondioxide→**

**registerTimedReportCallback( )**

---

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

**int registerTimedReportCallback( YCarbonDioxideTimedReportCallback callback)**

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

**Paramètres :**

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

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

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

```
int registerValueCallback( YCarbonDioxideValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

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

**carbondioxide→set\_highestValue()**

**YCarbonDioxide**

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

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double 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.

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

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

**int set\_logFrequency( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

---

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

---

Modifie le nom logique du capteur de CO2.

```
int set_logicalName( const string& newval)
```

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

**Paramètres :**

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

**Retourne :**

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

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

**YCarbonDioxide**

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

---

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

---

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**carbondioxide→set(userData)****YCarbonDioxide****carbondioxide→setUserData()carbon dioxide→  
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).

```
void set(userData void* 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 ColorLed

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_colorled.js'></script>
nodejs var yoctolib = require('yoctolib');
var YColorLed = yoctolib.YColorLed;
php require_once('yocto_colorled.php');
cpp #include "yocto_colorled.h"
m #import "yocto_colorled.h"
pas uses yocto_colorled;
vb yocto_colorled.vb
cs yocto_colorled.cs
java import com.yoctopuce.YoctoAPI.YColorLed;
py 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 de manière non-ambigüe l'instance de la led RGB au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### colorled→get\_advertisedValue()

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

#### colorled→get\_errorMessage()

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

#### colorled→get\_errorType()

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

#### colorled→get\_friendlyName()

Retourne un identifiant global de la led RGB au format NOM\_MODULE . NOM\_FONCTION.

#### colorled→get\_functionDescriptor()

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

#### colorled→get\_functionId()

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

#### colorled→get\_hardwareId()

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

#### colorled→get\_hslColor()

Retourne la couleur HSL courante de la led.

#### colorled→get\_logicalName()

Retourne le nom logique de la led RGB.

<b>colorled→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>colorled→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>colorled→get_rgbColor()</b>	Retourne la couleur RGB courante de la led.
<b>colorled→get_rgbColorAtPowerOn()</b>	Retourne la couleur configurée pour être affichage à l'allumage du module.
<b>colorled→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>colorled→hslMove(hsl_target, ms_duration)</b>	Effectue une transition continue dans l'espace HSL entre la couleur courante et une nouvelle couleur.
<b>colorled→isOnline()</b>	Vérifie si le module hébergeant la led RGB est joignable, sans déclencher d'erreur.
<b>colorled→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant la led RGB est joignable, sans déclencher d'erreur.
<b>colorled→load(msValidity)</b>	Met en cache les valeurs courantes de la led RGB, avec une durée de validité spécifiée.
<b>colorled→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de la led RGB, avec une durée de validité spécifiée.
<b>colorled→nextColorLed()</b>	Continue l'énumération des leds RGB commencée à l'aide de yFirstColorLed( ).
<b>colorled→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>colorled→rgbMove(rgb_target, ms_duration)</b>	Effectue une transition continue dans l'espace RGB entre la couleur courante et une nouvelle couleur.
<b>colorled→set_hslColor(newval)</b>	Modifie la couleur courante de la led, en utilisant une couleur HSL spécifiée.
<b>colorled→set_logicalName(newval)</b>	Modifie le nom logique de la led RGB.
<b>colorled→set_rgbColor(newval)</b>	Modifie la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu).
<b>colorled→set_rgbColorAtPowerOn(newval)</b>	Modifie la couleur que la led va afficher spontanément à l'allumage du module.
<b>colorled→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>colorled→wait_async(callback, context)</b>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

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

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

**YColorLed\* yFindColorLed( const string& 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()****yFirstColorLed()yFirstColorLed( )****YColorLed**

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

**YColorLed\* yFirstColorLed( )**

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()****YColorLed**

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

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant la led RGB (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

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

**YColorLed**

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

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

**YColorLed**

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

**string getErrorMessage( )**

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

**Retourne :**

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

**colorled→get\_errorType()****YColorLed****colorled→errorType()colorled→get\_errorType( )**

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

**YRETCODE get\_errorType( )**

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

**Retourne :**

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

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

**YColorLed**

Retourne un identifiant global de la led RGB au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

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

**Retourne :**

une chaîne de caractères identifiant la led RGB en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

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

**YColorLed**

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

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**colorled→get\_functionId()** YColorLed  
**colorled→functionId()** **colorled→get\_functionId( )**

---

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

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

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

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

**YColorLed**

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

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led RGB (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la led RGB (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**colorled→get\_hslColor()**

**YColorLed**

**colorled→hslColor()colorled→get\_hslColor()**

---

Retourne la couleur HSL courante de la led.

```
int 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()**  
**colorled→logicalName()colorled→**  
**get\_logicalName( )**

**YColorLed**

Retourne le nom logique de la led RGB.

```
string get_logicalName( )
```

**Retourne :**

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

**colorled→get\_module()**

**YColorLed**

**colorled→module()colorled→get\_module()**

---

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

**YModule \* get\_module( )**

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

**Retourne :**

une instance de **YModule**

**colorled→get\_rgbColor()****YColorLed****colorled→rgbColor()colorled→get\_rgbColor( )**

Retourne la couleur RGB courante de la led.

```
int get_rgbColor( )
```

**Retourne :**

un entier représentant la couleur RGB courante de la led

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

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

---

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

**int get\_rgbColorAtPowerOn( )**

**Retourne :**

un entier représentant la couleur configurée pour être affichage à l'allumage du module

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

**colorled→get(userData)****YColorLed****colorled→userData()colorled→get(userData())**

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

```
void * 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→hslMove()****YColorLed**

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

```
int hslMove( int hsl_target, int 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()****YColorLed**

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

**bool isOnline( )**

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

**Retourne :**

true si la led RGB est joignable, false sinon

**colorled→load()****YColorLed**

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

**YRETCODE load( int msValidity)**

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

**Paramètres :**

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

**Retourne :**

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

**colorled**→**nextColorLed()**  
**colorled**→  
**nextColorLed( )**

**YColorLed**

Continue l'énumération des leds RGB commencée à l'aide de **yFirstColorLed( )**.

**YColorLed \* nextColorLed( )**

**Retourne :**

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

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

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

```
int registerValueCallback( YColorLedValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

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

**colorled→rgbMove()****YColorLed**

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

```
int rgbMove( int rgb_target, int ms_duration)
```

**Paramètres :**

**rgb\_target** couleur RGB désirée à la fin de la transition

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

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

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

**colorled→set\_hslColor()**

**YColorLed**

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

---

Modifie la couleur courante de la led, en utilisant une couleur HSL spécifiée.

```
int set_hslColor( int newval)
```

L'encodage est réalisé de la manière suivante: 0xHHSSLL.

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

**YColorLed**

Modifie le nom logique de la led RGB.

**int set\_logicalName( const string& newval)**

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

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la led RGB.

**Retourne :**

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

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

**YColorLed**

---

Modifie la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu).

**int set\_rgbColor( int newval)**

L'encodage est réalisé de la manière suivante: 0xRRGGBB.

**Paramètres :**

**newval** un entier représentant la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu)

**Retourne :**

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

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

**colorled→set\_rgbColorAtPowerOn()**  
**colorled→setRgbColorAtPowerOn()colorled→**  
**set\_rgbColorAtPowerOn( )**

**YColorLed**

Modifie la couleur que la led va afficher spontanément à l'allumage du module.

**int set\_rgbColorAtPowerOn( int newval)**

Cette couleur sera affichée dès que le module sera sous tension. Ne pas oublier d'appeler la fonction `saveToFlash()` du module correspondant pour que ce paramètre soit mémorisé.

**Paramètres :**

**newval** un entier représentant la couleur que la led va afficher spontanément à l'allumage du module

**Retourne :**

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

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

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

**YColorLed**

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).

**void set(userData( void\* 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 Compass

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### **yFindCompass(func)**

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

#### **yFirstCompass()**

Commence l'énumération des compas accessibles par la librairie.

### Méthodes des objets YCompass

#### **compass→calibrateFromPoints(rawValues, refValues)**

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

#### **compass→describe()**

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

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

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

#### **compass→get\_currentRawValue()**

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

#### **compass→get\_currentValue()**

Retourne la valeur actuelle du cap relatif.

#### **compass→get\_errorMessage()**

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

#### **compass→get\_errorType()**

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

#### **compass→get\_friendlyName()**

Retourne un identifiant global du compas au format NOM\_MODULE .NOM\_FONCTION.

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

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

#### **compass→get\_functionId()**

Retourne l'identifiant matériel du compas, sans référence au module.

#### **compass→get\_hardwareId()**

Retourne l'identifiant matériel unique du compas au format SERIAL .FUNCTIONID.

### 3. Reference

<b>compass→get_highestValue()</b>	Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.
<b>compass→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>compass→get_logicalName()</b>	Retourne le nom logique du compas.
<b>compass→get_lowestValue()</b>	Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.
<b>compass→get_magneticHeading()</b>	Retourne la direction du nord magnétique, indépendamment du cap configuré.
<b>compass→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>compass→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>compass→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>compass→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>compass→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>compass→get_unit()</b>	Retourne l'unité dans laquelle le cap relatif est exprimée.
<b>compass→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>compass→isOnline()</b>	Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.
<b>compass→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.
<b>compass→load(msValidity)</b>	Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.
<b>compass→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>compass→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.
<b>compass→nextCompass()</b>	Continue l'énumération des compas commencée à l'aide de yFirstCompass( ).
<b>compass→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>compass→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>compass→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.

**compass→set\_logFrequency(newval)**

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

**compass→set\_logicalName(newval)**

Modifie le nom logique du compas.

**compass→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**compass→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**compass→set\_userData(data)**

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

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

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

## YCompass.FindCompass() yFindCompass()yFindCompass( )

**YCompass**

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

**YCompass\* yFindCompass( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

**func** une chaîne de caractères qui référence le compas sans ambiguïté

### Retourne :

un objet de classe `YCompass` qui permet ensuite de contrôler le compas.

**YCompass.FirstCompass()****YCompass****yFirstCompass()yFirstCompass( )**

Commence l'énumération des compas accessibles par la librairie.

**YCompass\* yFirstCompass( )**

Utiliser la fonction `YCompass.nextCompass()` pour itérer sur les autres compas.

**Retourne :**

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

**compass→calibrateFromPoints()**  
**compass→calibrateFromPoints( )****YCompass**

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

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

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

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

**Retourne :**

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

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

**compass->describe()****YCompass**

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

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le compas (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

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

---

**YCompass**

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

**string get\_advertisedValue( )**

**Retourne :**

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

**compass→get\_currentRawValue()**  
**compass→currentRawValue()**compass→  
**get\_currentRawValue( )**

**YCompass**

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

**double get\_currentRawValue( )**

**Retourne :**

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

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

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

---

**YCompass**

Retourne la valeur actuelle du cap relatif.

double **get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur actuelle du cap relatif

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

**compass→getErrorMessage()**  
**compass→errorMessage()compass→**  
**getErrorMessage( )**

**YCompass**

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

**string getErrorMessage( )**

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

**Retourne :**

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

**compass→get\_errorType()**

**YCompass**

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

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

**YRETCODE get\_errorType( )**

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

**Retourne :**

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

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

**YCompass**

Retourne un identifiant global du compas au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

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

**Retourne :**

une chaîne de caractères identifiant le compas en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

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

---

**YCompass**

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

**YFUN\_DESCR get\_functionDescriptor( )**

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

**Retourne :**

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

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

**YCompass**

Retourne l'identifiant matériel du compas, sans référence au module.

**string get\_functionId( )**

Par exemple relay1.

**Retourne :**

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

**compass→get\_hardwareId()**

**YCompass**

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

---

Retourne l'identifiant matériel unique du compas au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du compas (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le compas (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

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

**YCompass**

Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.

**double get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le cap relatif depuis le démarrage du module

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

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

**YCompass**

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

string **get\_logFrequency( )**

**Retourne :**

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

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

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

**YCompass**

Retourne le nom logique du compas.

**string get\_logicalName( )**

**Retourne :**

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

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

---

**YCompass**

Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.

double **get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le cap relatif depuis le démarrage du module

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

**compass→get\_magneticHeading()**  
**compass→magneticHeading()compass→**  
**get\_magneticHeading( )**

**YCompass**

Retourne la direction du nord magnétique, indépendemment du cap configuré.

**double get\_magneticHeading( )**

**Retourne :**

une valeur numérique représentant la direction du nord magnétique, indépendemment du cap configuré

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

**compass→get\_module()**

**YCompass**

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

---

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

**YModule \* get\_module( )**

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

**Retourne :**

une instance de YModule

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

**YCompass**

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

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

**compass→get\_reportFrequency()**  
**compass→reportFrequency()compass→**  
**get\_reportFrequency( )**

**YCompass**

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

string **get\_reportFrequency( )**

**Retourne :**

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

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

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

**YCompass**

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

**compass→get\_unit()**

**YCompass**

**compass→unit()compass→get\_unit()**

---

Retourne l'unité dans laquelle le cap relatif est exprimée.

string **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le cap relatif est exprimée

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

**compass→get(userData)****YCompass****compass→userData()compass→get(userData()**

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

```
void * 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.

## compass→isOnline()compass→isOnline( )

## YCompass

---

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

**bool isOnline( )**

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

**Retourne :**

true si le compas est joignable, false sinon

**compass→load()****YCompass**

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

**YRETCODE load( int msValidity)**

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

**Paramètres :**

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

**Retourne :**

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

**compass→loadCalibrationPoints()**  
**compass→  
loadCalibrationPoints( )****YCompass**

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

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

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

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

**Retourne :**

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

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

**compass→nextCompass()** compass→  
**nextCompass( )**

**YCompass**

Continue l'énumération des compas commencée à l'aide de `yFirstCompass()`.

`YCompass * nextCompass( )`

**Retourne :**

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

**compass→registerTimedReportCallback()** compass→  
**registerTimedReportCallback( )**

**YCompass**

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

```
int registerTimedReportCallback( YCompassTimedReportCallback callback)
```

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

**Paramètres :**

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

**compass→registerValueCallback()compass→registerValueCallback( )****YCompass**

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

```
int registerValueCallback( YCompassValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

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

**compass→set\_highestValue()**

**YCompass**

**compass→setHighestValue()compass→**

**set\_highestValue( )**

---

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

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

**Retourne :**

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

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

**compass→set\_logFrequency()**  
**compass→setLogFrequency()compass→**  
**set\_logFrequency( )**

**YCompass**

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

**int set\_logFrequency( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie le nom logique du compas.

**int set\_logicalName( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**YCompass**

Modifie la mémoire de valeur minimale observée.

int **set\_lowestValue( double newval)**

**Paramètres :**

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

**Retourne :**

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

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

**compass→set\_reportFrequency()** **YCompass**  
**compass→setReportFrequency()** **compass→**  
**set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

**compass→set(userData)**  
**compass→setUserData()** **compass→**  
**set(userData)**

**YCompass**

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).

**void set(userData void\* 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. Interface de la fonction Current

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

### Fonction globales

#### yFindCurrent(func)

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

#### yFirstCurrent()

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

### Méthodes des objets YCurrent

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

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

#### current→describe()

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

#### current→get\_advertisedValue()

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

#### current→get\_currentRawValue()

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

#### current→get\_currentValue()

Retourne la valeur instantanée du courant.

#### current→get\_errorMessage()

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

#### current→get\_errorType()

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

#### current→get\_friendlyName()

Retourne un identifiant global du capteur de courant au format NOM\_MODULE . NOM\_FONCTION.

#### current→get\_functionDescriptor()

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

#### current→get\_functionId()

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

#### current→get\_hardwareId()

### 3. Reference

Retourne l'identifiant matériel unique du capteur de courant au format SERIAL.FUNCTIONID.
<b>current→get_highestValue()</b> Retourne la valeur maximale observée pour le courant.
<b>current→get_logFrequency()</b> Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>current→get_logicalName()</b> Retourne le nom logique du capteur de courant.
<b>current→get_lowestValue()</b> Retourne la valeur minimale observée pour le courant.
<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_recordedData(startTime, endTime)</b> Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>current→get_reportFrequency()</b> Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>current→get_resolution()</b> Retourne la résolution des valeurs mesurées.
<b>current→get_unit()</b> Retourne l'unité dans laquelle le courant 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 le capteur de courant est joignable, sans déclencher d'erreur.
<b>current→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.
<b>current→load(msValidity)</b> Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.
<b>current→loadCalibrationPoints(rawValues, refValues)</b> Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>current→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du capteur de courant, 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→registerTimedReportCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<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 pour le courant.
<b>current→set_logFrequency(newval)</b>

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

**current→set\_logicalName(newval)**

Modifie le nom logique du capteur de courant.

**current→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour le courant.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**current→set\_resolution(newval)**

Modifie la résolution des valeurs mesurées.

**current→set\_userData(data)**

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

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

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

## YCurrent.FindCurrent() yFindCurrent()yFindCurrent( )

**YCurrent**

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

**YCurrent\* yFindCurrent( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

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

### Retourne :

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

**YCurrent.FirstCurrent()****YCurrent****yFirstCurrent()yFirstCurrent( )**

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

**YCurrent\* yFirstCurrent( )**

Utiliser la fonction `YCurrent.nextCurrent( )` pour itérer sur les autres capteurs de courant.

**Retourne :**

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

**current→calibrateFromPoints()current→calibrateFromPoints( )****YCurrent**

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

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**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()****YCurrent**

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

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de courant (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

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

---

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

string **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()**  
**current→currentRawValue()** current→  
**get\_currentRawValue( )**

**YCurrent**

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

**double get\_currentRawValue( )**

**Retourne :**

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

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

**current→get\_currentValue()**

**YCurrent**

**current→currentValue()** current→

**get\_currentValue( )**

---

Retourne la valeur instantanée du courant.

double **get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur instantanée du courant

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

---

**current→getErrorMessage()**  
**current→errorMessage()current→**  
**getErrorMessage( )**

**YCurrent**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
string getErrorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

**current→get\_errorType()**

**YCurrent**

**current→errorType()current→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

**current→get\_friendlyName()**  
**current→friendlyName()** current→  
**get\_friendlyName( )**

**YCurrent**

Retourne un identifiant global du capteur de courant au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de courant si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de courant (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de courant en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

<b>current→get_functionDescriptor()</b>	<b>YCurrent</b>
<b>current→functionDescriptor()current→get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**current→get\_functionId()****YCurrent****current→functionId()current→get\_functionId( )**

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de courant (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**current→get\_hardwareId()**

**YCurrent**

**current→hardwareId()current→get\_hardwareId( )**

---

Retourne l'identifiant matériel unique du capteur de courant au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de courant (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de courant (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**current→get\_highestValue()**  
**current→highestValue()current→**  
**get\_highestValue()**

**YCurrent**

Retourne la valeur maximale observée pour le courant.

**double get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le courant

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**current→get\_logFrequency()**  
**current→logFrequency()current→**  
**get\_logFrequency( )**

**YCurrent**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**current→get\_logicalName()**  
**current→logicalName()current→**  
**get\_logicalName( )**

**YCurrent**

Retourne le nom logique du capteur de courant.

```
string 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()**  
**current→lowestValue()** current→  
**get\_lowestValue()**

---

**YCurrent**

Retourne la valeur minimale observée pour le courant.

```
double get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le courant

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**current→get\_module()****YCurrent****current→module()current→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**une instance de `YModule`

**current→get\_recordedData()**  
**current→recordedData()current→**  
**get\_recordedData( )**

**YCurrent**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**current→get\_reportFrequency()**  
**current→reportFrequency()current→**  
**get\_reportFrequency( )**

**YCurrent**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**string get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**current→get\_resolution()**

**YCurrent**

**current→resolution()current→get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**current→get\_unit()****YCurrent****current→unit()current→get\_unit()**

---

Retourne l'unité dans laquelle le courant est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le courant est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**current→get(userData)**

**YCurrent**

**current→userData()current→get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**current→isOnline()****YCurrent**

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de courant est joignable, false sinon

**current→load()****YCurrent**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→loadCalibrationPoints()**  
**current→  
loadCalibrationPoints( )****YCurrent**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→nextCurrent()current→nextCurrent( )**

**YCurrent**

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

`YCurrent * nextCurrent( )`

**Retourne :**

un pointeur sur un objet `YCurrent` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**current→registerTimedReportCallback()**  
**current→registerTimedReportCallback( )****YCurrent**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YCurrentTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**current→registerValueCallback()****current→registerValueCallback( )**

**YCurrent**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**int registerValueCallback( YCurrentValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**current→set\_highestValue()**  
**current→setHighestValue()** current→  
**set\_highestValue()**

**YCurrent**

Modifie la mémoire de valeur maximale observée pour le courant.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour le courant

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_logFrequency()**  
**current→setLogFrequency()** current→  
**set\_logFrequency( )**

**YCurrent**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_logicalName()**  
**current→setLogicalName()** current→  
**set\_logicalName( )**

**YCurrent**

Modifie le nom logique du capteur de courant.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser **yCheckLogicalName( )** pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode **saveToFlash( )** du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de courant.

**Retourne :**

**YAPI\_SUCCESS** si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_lowestValue()** YCurrent  
**current→setLowestValue()** current→  
**set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour le courant.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour le courant

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_reportFrequency()**  
**current→setReportFrequency()current→**  
**set\_reportFrequency( )**

**YCurrent**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_resolution()**  
**current→setResolution()** current→  
**set\_resolution()**

**YCurrent**

Modifie la résolution des valeurs mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set(userData)****YCurrent****current→setUserData()current→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :****data** objet quelconque à mémoriser

## 3.8. Interface de la fonction DataLogger

Les capteurs de Yoctopuce sont équipés d'une mémoire non-volatile permettant de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La fonction DataLogger contrôle les paramètres globaux de cet enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_datalogger.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDataLogger = yoctolib.YDataLogger;
require_once('yocto_datalogger.php');
php #include "yocto_datalogger.h"
cpp #import "yocto_datalogger.h"
m uses yocto_datalogger;
pas yocto_datalogger.vb
cs yocto_datalogger.cs
java import com.yoctopuce.YoctoAPI.YDataLogger;
py from yocto_datalogger import *

```

### Fonction globales

#### **yFindDataLogger(func)**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

#### **yFirstDataLogger()**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

### Méthodes des objets YDataLogger

#### **datalogger→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **datalogger→forgetAllDataStreams()**

Efface tout l'historique des mesures de l'enregistreur de données.

#### **datalogger→get\_advertisedValue()**

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

#### **datalogger→get\_autoStart()**

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

#### **datalogger→get\_currentRunIndex()**

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

#### **datalogger→get\_dataSets()**

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

#### **datalogger→get\_dataStreams(*v*)**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

#### **datalogger→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

#### **datalogger→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

#### **datalogger→get\_friendlyName()**

Retourne un identifiant global de l'enregistreur de données au format NOM\_MODULE . NOM\_FONCTION.

#### **datalogger→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **datalogger→get\_functionId()**

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

#### **datalogger→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'enregistreur de données au format SERIAL . FUNCTIONID.

#### **datalogger→get\_logicalName()**

Retourne le nom logique de 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\_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 l'enregistreur de données est joignable, sans déclencher d'erreur.

#### **datalogger→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

#### **datalogger→load(msValidity)**

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

#### **datalogger→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'enregistreur de données, 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).

#### **datalogger→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YDataLogger.FindDataLogger()****YDataLogger****yFindDataLogger()yFindDataLogger( )**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

**YDataLogger\* yFindDataLogger( string 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()****yFirstDataLogger()yFirstDataLogger( )****YDataLogger**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

**YDataLogger\* yFirstDataLogger( )**

Utiliser la fonction `YDataLogger.nextDataLogger( )` pour itérer sur les autres enregistreurs de données.

**Retourne :**

un pointeur sur un objet `YDataLogger`, correspondant au premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

**datalogger→describe()****YDataLogger**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'enregistreur de données (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**datalogger→forgetAllDataStreams()datalogger→  
forgetAllDataStreams( )****YDataLogger**

Efface tout l'historique des mesures de l'enregistreur de données.

```
int 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()**  
**datalogger→advertisedValue()datalogger→**  
**get\_advertisedValue( )**

---

**YDataLogger**

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

**string 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()**  
**datalogger→autoStart()datalogger→**  
**get\_autoStart( )**

**YDataLogger**

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

**Y\_AUTOSTART\_enum 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()**  
**datalogger→currentRunIndex()datalogger→**  
**get\_currentRunIndex( )**

**YDataLogger**

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.

**int get\_currentRunIndex( )**

**Retourne :**

un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRUNINDEX\_INVALID.

**datalogger→get\_dataSets()**  
**datalogger→dataSets()datalogger→**  
**get\_dataSets( )**

**YDataLogger**

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

`vector<YDataSet> get_dataSets( )`

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Retourne :**

une liste d'objets YDataSet

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**datalogger→get\_dataStreams()**  
**datalogger→dataStreams()datalogger→**  
**get\_dataStreams( )**

**YDataLogger**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

**int get\_dataStreams( )**

L'appelant doit passer par référence un tableau vide pour stocker les objets YDataStream, et la méthode va les remplir avec des objets décrivant les séquences de données disponibles.

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

**Paramètres :**

- ✓ 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()**  
**datalogger→errorMessage()datalogger→**  
**get\_errorMessage( )**

**YDataLogger**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

```
string get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

**datalogger→get\_errorType()**  
**datalogger→errorType()****datalogger→get\_errorType( )**

---

**YDataLogger**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

**datalogger→get\_friendlyName()****YDataLogger****datalogger→friendlyName()datalogger→  
get\_friendlyName( )**

Retourne un identifiant global de l'enregistreur de données au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne retournée utilise soit les noms logiques du module et de l'enregistreur de données si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'enregistreur de données (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**datalogger→get\_functionDescriptor()**  
**datalogger→functionDescriptor()datalogger→**  
**get\_functionDescriptor( )**

---

**YDataLogger**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**datalogger→get\_functionId()**  
**datalogger→functionId()datalogger→**  
**get\_functionId( )**

**YDataLogger**

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**datalogger→get\_hardwareId()**  
**datalogger→hardwareId()****datalogger→get\_hardwareId( )**

**YDataLogger**

Retourne l'identifiant matériel unique de l'enregistreur de données au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'enregistreur de données (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**datalogger→get\_logicalName()**  
**datalogger→logicalName()datalogger→**  
**get\_logicalName( )**

**YDataLogger**

Retourne le nom logique de l'enregistreur de données.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique de l'enregistreur de données. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**datalogger→get\_module()**

**YDataLogger**

**datalogger→module()datalogger→get\_module( )**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**datalogger→get\_recording()**  
**datalogger→recording()datalogger→**  
**get\_recording( )**

**YDataLogger**

Retourne l'état d'activation de l'enregistreur de données.

**Y\_RECORDING\_enum 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()**  
**datalogger→timeUTC()****datalogger→get\_timeUTC( )**

---

**YDataLogger**

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

s64 **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)**  
**datalogger→userData()datalogger→**  
**get(userData( )**

**YDataLogger**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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()****YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'enregistreur de données est joignable, false sinon

**datalogger→load()****YDataLogger**

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→nextDataLogger()**  
**datalogger→**  
**nextDataLogger( )**

**YDataLogger**

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

**YDataLogger \* nextDataLogger( )**

**Retourne :**

un pointeur sur un objet `YDataLogger` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**datalogger→registerValueCallback()**  
**datalogger→registerValueCallback( )****YDataLogger**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YDataLoggerValueCallback 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 appellés 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()**  
**datalogger→setAutoStart()****datalogger→set\_autoStart( )**

**YDataLogger**

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

**int set\_autoStart( Y\_AUTOSTART\_enum 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()**  
**datalogger→setLogicalName()****datalogger→set\_logicalName( )**

**YDataLogger**

Modifie le nom logique de l'enregistreur de données.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'enregistreur de données.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→set\_recording()**  
**datalogger→setRecording()****datalogger→set\_recording( )**

**YDataLogger**

Modifie l'état d'activation de l'enregistreur de données.

**int set\_recording( Y\_RECORDING\_enum 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()**  
**datalogger→setTimeUTC()****datalogger→set\_timeUTC()**

**YDataLogger**

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

**int set\_timeUTC( s64 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)**  
**datalogger→setUserData()****datalogger→set(userData)**

**YDataLogger**

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).

**void set(userData void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.9. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_datalogger.js'></script>
node.js var yoctolib = require('yoctolib');
          var YDataLogger = yoctolib.YDataLogger;
php require_once('yocto_datalogger.php');
cpp #include "yocto_datalogger.h"
m #import "yocto_datalogger.h"
pas uses yocto_datalogger;
vb yocto_datalogger.vb
cs yocto_datalogger.cs
java import com.yoctopuce.YoctoAPI.YDataLogger;
py from yocto_datalogger import *

```

### 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\_startTimeUTC()**  
**datarun→startTimeUTC()**

---

**YDataRun**

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.

## 3.10. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-mêmes sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

Cette classe ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Méthodes des objets YDataSet

#### `dataset→get_endTimeUTC()`

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### `dataset→get_functionId()`

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

#### `dataset→get_hardwareId()`

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

#### `dataset→get_measures()`

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

#### `dataset→get_preview()`

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

#### `dataset→get_progress()`

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

#### `dataset→get_startTimeUTC()`

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### `dataset→get_summary()`

Retourne un objet YMeasure résumant tout le YDataSet.

#### `dataset→get_unit()`

### **3. Reference**

---

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

#### **dataset→loadMore()**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

#### **dataset→loadMore\_async(callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**dataset→get\_endTimeUTC()**  
**dataset→endTimeUTC()dataset→**  
**get\_endTimeUTC( )**

**YDataSet**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

**s64 get\_endTimeUTC( )**

Lorsque l'objet YDataSet est créé, l'heure de fin est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore( )`, l'heure de fin est mise à jour à la dernière mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la dernière mesure.

**dataset→get\_functionId()**

**YDataSet**

**dataset→functionId()dataset→get\_functionId()**

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

string **get\_functionId( )**

Par exemple `temperature1`.

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: `temperature1`)

**dataset→get\_hardwareId()****YDataSet****dataset→hardwareId()dataset→get\_hardwareId( )**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple THRMCPL1-123456.temperature1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: THRMCPL1-123456.temperature1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**dataset→get\_measures()****YDataSet****dataset→measures()dataset→get\_measures( )**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

**vector<YMeasure> get\_measures( )**

Chaque élément contient: - le moment où la mesure a débuté - le moment où la mesure s'est terminée - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Avant d'appeler cette méthode, vous devez appeler `loadMore( )` pour charger des données depuis l'enregistreur sur le module. L'appel doit être répété plusieurs fois pour charger toutes les données, mais vous pouvez commencer à utiliser les données disponibles avant qu'elles n'aient été toutes chargées

Les mesures les plus anciennes sont toujours chargées les premières, et les plus récentes en dernier. De ce fait, les timestamps dans la table des mesures sont normalement par ordre chronologique. La seule exception est dans le cas où il y a eu un ajustement de l'horloge UTC de l'enregistreur de données pendant l'enregistrement.

**Retourne :**

un tableau d'enregistrements, chaque enregistrement représentant une mesure effectuée à un moment précis.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**dataset→get\_preview()****YDataSet****dataset→preview()dataset→get\_preview( )**

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

```
vector<YMeasure> get_preview( )
```

Chaque élément contient: - le début d'un intervalle de temps - la fin d'un intervalle de temps - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Le résumé des mesures est disponible dès que `loadMore( )` a été appelé pour la première fois.

**Retourne :**

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**dataset→get\_progress()**

**YDataSet**

**dataset→progress()dataset→get\_progress( )**

---

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

**int get\_progress( )**

A l'instanciation de l'objet par la fonction `get_dataSet( )`, l'avancement est nul. Au fur et à mesure des appels à `loadMore( )`, l'avancement progresse pour atteindre la valeur 100 lorsque toutes les mesures ont été chargées.

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées.

---

**dataset→getStartTimeUTC()**  
**dataset→startUTC()dataset→**  
**getStartTimeUTC()**

**YDataSet**

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

s64 **getStartTimeUTC( )**

Lorsque l'objet YDataSet est créé, l'heure de départ est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore( )`, l'heure de départ est mise à jour à la première mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la première mesure enregistrée.

**dataset→get\_summary()**

**YDataSet**

**dataset→summary()dataset→get\_summary( )**

---

Retourne un objet YMeasure résumant tout le YDataSet.

**YMeasure get\_summary( )**

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `loadMore( )` a été appelé pour la première fois.

**Retourne :**

un objet YMeasure

---

**dataset→get\_unit()****YDataSet****dataset→unit()dataset→get\_unit()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant une unité physique.

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

## dataset→loadMore() dataset→loadMore( )

## YDataSet

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

```
int loadMore( )
```

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.11. Séquence de données enregistrées brute

Les objets YDataStream correspondent aux séquences de mesures enregistrées brutes, directement telles qu'obtenues par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Dans la plupart des cas, il n'est pas nécessaire d'utiliser les objets DataStream, car les objets YDataSet (retournés par la méthode `get_recordedData()` des senseurs et la méthode `get_dataSets()` du DataLogger) fournissent une interface plus pratique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
node.js var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Méthodes des objets YDataStream

#### `datastream→get_averageValue()`

Retourne la moyenne des valeurs observées durant cette séquence.

#### `datastream→get_columnCount()`

Retourne le nombre de colonnes de données contenus dans la séquence.

#### `datastream→get_columnNames()`

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

#### `datastream→get_data(row, col)`

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

#### `datastream→get_dataRows()`

Retourne toutes les données mesurées contenus dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

#### `datastream→get_dataSamplesIntervalMs()`

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

#### `datastream→get_duration()`

Retourne la durée approximative de cette séquence, en secondes.

#### `datastream→get_maxValue()`

Retourne la plus grande valeur observée durant cette séquence.

#### `datastream→get_minValue()`

Retourne la plus petite valeur observée durant cette séquence.

#### `datastream→getRowCount()`

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()`

### **3. Reference**

---

Retourne le temps de départ relatif de la séquence (en secondes).

#### **datastream→get\_startTimeUTC()**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

**datastream→get\_averageValue()**  
**datastream→averageValue()datastream→**  
**get\_averageValue()**

**YDataStream**

Retourne la moyenne des valeurs observées durant cette séquence.

**double get\_averageValue( )**

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la moyenne des valeurs, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→get\_columnCount()**  
**datastream→columnCount()datastream→**  
**get\_columnCount( )**

**YDataStream**

Retourne le nombre de colonnes de données contenus dans la séquence.

**int get\_columnCount( )**

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames( )`.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclanche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre de colonnes.

En cas d'erreur, déclenche une exception ou retourne zéro.

**datastream→get\_columnNames()**  
**datastream→columnNames()****datastream→get\_columnNames( )**

**YDataStream**

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

**vector<string> get\_columnNames( )**

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure. Pour les séquences enregistrées à faible fréquence, l'enregistreur de donnée stocke la valeur min, moyenne et max observée durant chaque intervalle de temps dans des colonnes avec les suffixes \_min, \_avg et \_max respectivement.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

une liste de chaîne de caractères.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**datastream→get\_data()** **YDataStream**  
**datastream→data()datastream→get\_data( )**

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

```
double get_data( int row, int col)
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche 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()**  
**datastream→dataRows()datastream→**  
**get\_dataRows( )**

**YDataStream**

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

```
vector< vector<double> > get_dataRows( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Retourne :**

une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**datastream→get\_dataSamplesIntervalMs()**  
**datastream→dataSamplesIntervalMs()datastream→**  
**get\_dataSamplesIntervalMs( )**

---

**YDataStream**

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

**int get\_dataSamplesIntervalMs( )**

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la fréquence d'enregistrement peut être changée pour chaque fonction.

**Retourne :**

un entier positif correspondant au nombre de millisecondes entre deux mesures consécutives.

**datastream→get\_duration()**  
**datastream→duration()datastream→**  
**get\_duration( )**

**YDataStream**

Retourne la durée approximative de cette séquence, en secondes.

`int get_duration( )`

**Retourne :**

le nombre de secondes couvertes par cette séquence.

En cas d'erreur, déclenche une exception ou retourne Y\_DURATION\_INVALID.

**datastream→get\_maxValue()**  
**datastream→maxValue()** **datastream→get\_maxValue( )**

---

**YDataStream**

Retourne la plus grande valeur observée durant cette séquence.

**double get\_maxValue( )**

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus grande valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→get\_minValue()**  
**datastream→minValue()** **datastream→get\_minValue( )**

**YDataStream**

Retourne la plus petite valeur observée durant cette séquence.

**double get\_minValue( )**

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus petite valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→getRowCount()**  
**datastream→rowCount()** **datastream→getRowCount()**

**YDataStream**

Retourne le nombre d'enregistrement contenus dans la séquence.

**int getRowCount()**

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclanche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre d'enregistrements.

En cas d'erreur, déclenche une exception ou retourne zéro.

**datastream→get\_runIndex()**

**YDataStream**

**datastream→runIndex()datastream→  
get\_runIndex( )**

---

Retourne le numéro de Run de la séquence de données.

```
int get_runIndex( )
```

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

**Retourne :**

un entier positif correspondant au numéro du Run

**datastream→getStartTime()**  
**datastream→startTime()datastream→**  
**getStartTime( )**

**YDataStream**

Retourne le temps de départ relatif de la séquence (en secondes).

**int getStartTime( )**

Pour les firmwares récents, la valeur est relative à l'heure courante (valeur négative). Pour les modules utilisant un firmware plus ancien que la version 13000, la valeur est le nombre de secondes depuis la mise sous tension du module (valeur positive). Si vous désirez obtenir l'heure absolue du début de la séquence, utilisez `getStartTimeUTC( )`.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le début du Run et le début de la séquence enregistrée.

**datastream→getStartTimeUTC()**  
**datastream→startTimeUTC()datastream→**  
**getStartTimeUTC()**

**YDataStream**

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).

s64 **getStartTimeUTC( )**

Si l'heure UTC n'était pas configurée dans l'enregistreur de données au début de la séquence, cette méthode retourne 0.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début de la séquence enregistrée.

## 3.12. Interface de la fonction DigitalIO

La librairie de programmation Yoctopuce permet simplement de changer l'état de chaque bit du port d'entrée sortie. Il est possible de changer tous les bits du port à la fois, ou de les changer indépendamment. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Le comportement électrique de chaque entrée/sortie peut être modifié (open drain et polarité inverse).

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_digitalio.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDigitalIO = yoctolib.YDigitalIO;
php require_once('yocto_digitalio.php');
cpp #include "yocto_digitalio.h"
m #import "yocto_digitalio.h"
pas uses yocto_digitalio;
vb yocto_digitalio.vb
cs yocto_digitalio.cs
java import com.yoctopuce.YoctoAPI.YDigitalIO;
py from yocto_digitalio import *

```

### Fonction globales

#### **yFindDigitalIO(func)**

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

#### **yFirstDigitalIO()**

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

### Méthodes des objets YDigitalIO

#### **digitalio→delayedPulse(bitno, ms\_delay, ms\_duration)**

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

#### **digitalio→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format TYPE (NAME )=SERIAL . FUNCTIONID.

#### **digitalio→get\_advertisedValue()**

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

#### **digitalio→get\_bitDirection(bitno)**

Retourne la direction d'un seul bit du port d'E/S.

#### **digitalio→get\_bitOpenDrain(bitno)**

Retourne la direction d'un seul bit du port d'E/S.

#### **digitalio→get\_bitPolarity(bitno)**

Retourne la polarité d'un seul bit du port d'E/S.

#### **digitalio→get\_bitState(bitno)**

Retourne l'état d'un seul bit du port d'E/S.

#### **digitalio→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

#### **digitalio→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

#### **digitalio→get\_friendlyName()**

Retourne un identifiant global du port d'E/S digital au format NOM\_MODULE . NOM\_FONCTION.

#### **digitalio→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **digitalio→get\_functionId()**

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

#### **digitalio→get\_hardwareId()**

Retourne l'identifiant matériel unique du port d'E/S digital au format SERIAL.FUNCTIONID.

#### **digitalio→get\_logicalName()**

Retourne le nom logique du port d'E/S digital.

#### **digitalio→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **digitalio→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **digitalio→get\_outputVoltage()**

Retourne la source de tension utilisée pour piloter les bits en sortie.

#### **digitalio→get\_portDirection()**

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

#### **digitalio→get\_portOpenDrain()**

Retourne le type d'interface électrique de chaque bit du port (bitmap).

#### **digitalio→get\_portPolarity()**

Retourne la polarité des bits du port (bitmap).

#### **digitalio→get\_portSize()**

Retourne le nombre de bits implémentés dans le port d'E/S.

#### **digitalio→get\_portState()**

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

#### **digitalio→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **digitalio→isOnline()**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

#### **digitalio→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

#### **digitalio→load(msValidity)**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

#### **digitalio→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

#### **digitalio→nextDigitalIO()**

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de yFirstDigitalIO().

#### **digitalio→pulse(bitno, ms\_duration)**

Déclenche une impulsion de durée spécifiée sur un bit choisi.

#### **digitalio→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **digitalio→set\_bitDirection(bitno, bitdirection)**

Change la direction d'un seul bit du port d'E/S.

#### **digitalio→set\_bitOpenDrain(bitno, opendrain)**

Change le type d'interface électrique d'un seul bit du port d'E/S.

#### **digitalio→set\_bitPolarity(bitno, bitpolarity)**

Change la polarité d'un seul bit du port d'E/S.

### 3. Reference

#### **digitalio→set\_bitState(bitno, bitstate)**

Change l'état d'un seul bit du port d'E/S.

#### **digitalio→set\_logicalName(newval)**

Modifie le nom logique du port d'E/S digital.

#### **digitalio→set\_outputVoltage(newval)**

Modifie la source de tension utilisée pour piloter les bits en sortie.

#### **digitalio→set\_portDirection(newval)**

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

#### **digitalio→set\_portOpenDrain(newval)**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

#### **digitalio→set\_portPolarity(newval)**

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

#### **digitalio→set\_portState(newval)**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

#### **digitalio→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **digitalio→toggle\_bitState(bitno)**

Inverse l'état d'un seul bit du port d'E/S.

#### **digitalio→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YDigitalIO.FindDigitalIO()****YDigitalIO****yFindDigitalIO()yFindDigitalIO( )**

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

**YDigitalIO\* yFindDigitalIO( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port d'E/S digital soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDigitalIO.isOnline()` pour tester si le port d'E/S digital est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le port d'E/S digital sans ambiguïté

**Retourne :**

un objet de classe `YDigitalIO` qui permet ensuite de contrôler le port d'E/S digital.

**YDigitalIO.FirstDigitalIO()**  
**yFirstDigitalIO()yFirstDigitalIO( )**

---

**YDigitalIO**

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

**YDigitalIO\* yFirstDigitalIO( )**

Utiliser la fonction YDigitalIO.nextDigitalIO( ) pour itérer sur les autres ports d'E/S digitaux.

**Retourne :**

un pointeur sur un objet YDigitalIO, correspondant au premier port d'E/S digital accessible en ligne, ou null si il n'y a pas de ports d'E/S digitaux disponibles.

**digitalio→delayedPulse()digitalio→  
delayedPulse()****YDigitalIO**

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

```
int delayedPulse( int bitno, int ms_delay, int ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**ms\_delay** délai d'attente avant l'impulsion, en millisecondes

**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→describe()digitalio→describe( )****YDigitalIO**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le port d'E/S digital (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**digitalio→get\_advertisedValue()**  
**digitalio→advertisedValue()digitalio→**  
**get\_advertisedValue( )**

**YDigitalIO**

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du port d'E/S digital (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**digitalio→get\_bitDirection()**  
**digitalio→bitDirection()digitalio→**  
**get\_bitDirection( )**

**YDigitalIO**

Retourne la direction d'un seul bit du port d'E/S.

**int get\_bitDirection( int bitno)**

(0 signifie que le bit est une entrée, 1 une sortie)

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→get\_bitOpenDrain()**  
**digitalio→bitOpenDrain()digitalio→**  
**get\_bitOpenDrain()**

**YDigitalIO**

Retourne la direction d'un seul bit du port d'E/S.

**int get\_bitOpenDrain( int bitno )**

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert)..

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**digitalio→get\_bitPolarity()** YDigitalIO  
**digitalio→bitPolarity()** **digitalio→get\_bitPolarity()**

---

Retourne la polarité d'un seul bit du port d'E/S.

**int get\_bitPolarity( int bitno )**

0 signifie que l'entrée sortie est en mode normal, 1 qu'elle est en mode inverse

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→get\_bitState()****YDigitalIO****digitalio→bitState()digitalio→get\_bitState( )**

Retourne l'état d'un seul bit du port d'E/S.

```
int get_bitState( int bitno)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

l'état du bit (0 ou 1).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→get\_errorMessage()**  
**digitalio→errorMessage()digitalio→**  
**get\_errorMessage( )**

**YDigitalIO**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

**digitalio→get\_errorType()**  
**digitalio→errorType()digitalio→**  
**get\_errorType( )**

**YDigitalIO**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

**digitalio→get\_friendlyName()**  
**digitalio→friendlyName()digitalio→**  
**get\_friendlyName( )**

---

**YDigitalIO**

Retourne un identifiant global du port d'E/S digital au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du port d'E/S digital si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port d'E/S digital (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**digitalio→get\_functionDescriptor()**  
**digitalio→functionDescriptor()digitalio→**  
**get\_functionDescriptor( )**

**YDigitalIO**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**digitalio→get\_functionId()**  
**digitalio→functionId()** digitalio→  
**get\_functionId( )**

---

**YDigitalIO**

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**digitalio→get.hardwareId()**  
**digitalio→hardwareId()** digitalio→  
**get.hardwareId( )**

**YDigitalIO**

Retourne l'identifiant matériel unique du port d'E/S digital au format SERIAL.FUNCTIONID.

**string get.hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port d'E/S digital (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**digitalio→get\_logicalName()**  
**digitalio→logicalName()digitalio→**  
**get\_logicalName( )**

---

**YDigitalIO**

Retourne le nom logique du port d'E/S digital.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du port d'E/S digital. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**digitalio→get\_module()****YDigitalIO****digitalio→module()digitalio→get\_module( )**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**une instance de `YModule`

**digitalio→get\_outputVoltage()**  
**digitalio→outputVoltage()digitalio→**  
**get\_outputVoltage()**

**YDigitalIO**

Retourne la source de tension utilisée pour piloter les bits en sortie.

**Y\_OUTPUTVOLTAGE\_enum get\_outputVoltage( )**

**Retourne :**

une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUTVOLTAGE\_INVALID.

**digitalio→get\_portDirection()**  
**digitalio→portDirection()digitalio→**  
**get\_portDirection()**

**YDigitalIO**

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

**int get\_portDirection( )**

**Retourne :**

un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

En cas d'erreur, déclenche une exception ou retourne Y\_PORTDIRECTION\_INVALID.

**digitalio→get\_portOpenDrain()**  
**digitalio→portOpenDrain()digitalio→**  
**get\_portOpenDrain( )**

---

**YDigitalIO**

Retourne le type d'interface électrique de chaque bit du port (bitmap).

**int get\_portOpenDrain( )**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

**Retourne :**

un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y\_PORTOPENDRAIN\_INVALID.

**digitalio→get\_portPolarity()**  
**digitalio→portPolarity()** digitalio→  
**get\_portPolarity()**

**YDigitalIO**

Retourne la polarité des bits du port (bitmap).

**int get\_portPolarity( )**

Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

**Retourne :**

un entier représentant la polarité des bits du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y\_PORTPOLARITY\_INVALID.

**digitalio→get\_portSize()**

**YDigitalIO**

**digitalio→portSize()digitalio→get\_portsize( )**

---

Retourne le nombre de bits implémentés dans le port d'E/S.

**int get\_portSize( )**

**Retourne :**

un entier représentant le nombre de bits implémentés dans le port d'E/S

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSIZE_INVALID`.

**digitalio→get\_portState()****YDigitalIO****digitalio→portState()digitalio→get\_portState( )**

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
int get_portState( )
```

**Retourne :**

un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

En cas d'erreur, déclenche une exception ou retourne Y\_PORTSTATE\_INVALID.

**digitalio→get(userData)**

**YDigitalIO**

**digitalio→userData()digitalio→get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**digitalio→isOnline()****YDigitalIO**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le port d'E/S digital est joignable, false sinon

**digitalio→load()****YDigitalIO**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→nextDigitalIO()digitalio→**  
**nextDigitalIO( )**

**YDigitalIO**

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

`YDigitalIO * nextDigitalIO( )`

**Retourne :**

un pointeur sur un objet `YDigitalIO` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**digitalio→pulse()digitalio→pulse( )****YDigitalIO**

Déclenche une impulsion de durée spécifiée sur un bit choisi.

```
int pulse( int bitno, int ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→registerValueCallback()  
digitalio→  
registerValueCallback( )****YDigitalIO**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YDigitalIOValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**digitalio→set\_bitDirection()**  
**digitalio→setBitDirection()digitalio→**  
**set\_bitDirection( )**

**YDigitalIO**

Change la direction d'un seul bit du port d'E/S.

**int set\_bitDirection( int bitno, int bitdirection)**

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitdirection** nouvelle valeur de la direction, 0=entrée, 1=sortie. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_bitOpenDrain()**  
**digitalio→setBitOpenDrain()** digitalio→  
**set\_bitOpenDrain()**

**YDigitalIO**

Change le type d'interface électrique d'un seul bit du port d'E/S.

```
int set_bitOpenDrain( int bitno, int opendrain)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**opendrain** 0 pour faire une entrée ou une sortie digitale standard, 1 pour une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_bitPolarity()**  
**digitalio→setBitPolarity()**  
**digitalio→set\_bitPolarity()**

**YDigitalIO**

Change la polarité d'un seul bit du port d'E/S.

**int set\_bitPolarity( int bitno, int bitpolarity)**

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitpolarity** nouvelle valeur de la polarité. 0=mode normal, 1=mode inverse. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_bitState()**  
**digitalio→setBitState()****digitalio→**  
**set\_bitState( )**

**YDigitalIO**

Change l'état d'un seul bit du port d'E/S.

```
int set_bitState( int bitno, int bitstate)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0  
**bitstate** nouvel état du bit (1 ou 0)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_logicalName()**  
**digitalio→setLogicalName()** digitalio→  
**set\_logicalName( )**

**YDigitalIO**

Modifie le nom logique du port d'E/S digital.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du port d'E/S digital.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_outputVoltage()**  
**digitalio→setOutputVoltage()digitalio→**  
**set\_outputVoltage( )**

**YDigitalIO**

Modifie la source de tension utilisée pour piloter les bits en sortie.

```
int set_outputVoltage( Y_OUTPUTVOLTAGE_enum newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé après un redémarrage du module.

**Paramètres :**

**newval** une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portDirection()**  
**digitalio→setPortDirection()****digitalio→**  
**set\_portDirection( )**

**YDigitalIO**

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

**int set\_portDirection( int newval)**

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portOpenDrain()**  
**digitalio→setPortOpenDrain()digitalio→**  
**set\_portOpenDrain()**

**YDigitalIO**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

**int set\_portOpenDrain( int newval)**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portPolarity()**  
**digitalio→setPortPolarity()**  
**digitalio→set\_portPolarity( )**

**YDigitalIO**

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne de manière inversée.

**int set\_portPolarity( int newval)**

**Paramètres :**

**newval** un entier représentant la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne de manière inversée

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portState()**  
**digitalio→setPortState()digitalio→**  
**set\_portState( )**

**YDigitalIO**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

**int set\_portState( int newval)**

Seuls les bits configurés en sortie dans portDirection sont affectés.

**Paramètres :**

**newval** un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set(userData())**  
**digitalio→setUserData()digitalio→**  
**set(userData( )**

**YDigitalIO**

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).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**digitalio→toggle\_bitState()digitalio→  
toggle\_bitState( )****YDigitalIO**

Inverse l'état d'un seul bit du port d'E/S.

```
int toggle_bitState( int bitno)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.13. Interface de la fonction Display

L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_display.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDisplay = yoctolib.YDisplay;
require_once('yocto_display.php');
cpp #include "yocto_display.h"
m #import "yocto_display.h"
pas uses yocto_display;
vb yocto_display.vb
cs yocto_display.cs
java import com.yoctopuce.YoctoAPI.YDisplay;
py from yocto_display import *

```

### Fonction globales

#### yFindDisplay(func)

Permet de retrouver un ecran d'après un identifiant donné.

#### yFirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

### Méthodes des objets YDisplay

#### display→copyLayerContent(srcLayerId, dstLayerId)

Copie le contenu d'un couche d'affichage vers une autre couche.

#### display→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format TYPE(NAME)=SERIAL.FUNCTIONID.

#### display→fade(brightness, duration)

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

#### display→get\_advertisedValue()

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

#### display→get\_brightness()

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

#### display→get\_displayHeight()

Retourne la hauteur de l'écran, en pixels.

#### display→get\_displayLayer(layerId)

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

#### display→get\_displayType()

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

#### display→get\_displayWidth()

Retourne la largeur de l'écran, en pixels.

#### display→get\_enabled()

Retourne vrai si le l'écran est alimenté, faux sinon.

#### display→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

#### **display→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

#### **display→get\_friendlyName()**

Retourne un identifiant global de l'écran au format NOM\_MODULE . NOM\_FONCTION.

#### **display→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **display→get\_functionId()**

Retourne l'identifiant matériel de l'écran, sans référence au module.

#### **display→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'écran au format SERIAL . FUNCTIONID.

#### **display→get\_layerCount()**

Retourne le nombre des couches affichables disponibles.

#### **display→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

#### **display→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

#### **display→get\_logicalName()**

Retourne le nom logique de l'écran.

#### **display→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **display→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **display→get\_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

#### **display→get\_startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

#### **display→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **display→isOnline()**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

#### **display→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

#### **display→load(msValidity)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

#### **display→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

#### **display→newSequence()**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

#### **display→nextDisplay()**

Continue l'énumération des écrans commencée à l'aide de yFirstDisplay( ).

#### **display→pauseSequence(delay\_ms)**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

**display→playSequence(sequenceName)**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes newSequence( ) et saveSequence( ).

**display→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**display→resetAll()**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

**display→saveSequence(sequenceName)**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

**display→set\_brightness(newval)**

Modifie la luminosité de l'écran.

**display→set\_enabled(newval)**

Modifie l'état d'activité de l'écran.

**display→set\_logicalName(newval)**

Modifie le nom logique de l'écran.

**display→set\_orientation(newval)**

Modifie l'orientation de l'écran.

**display→set\_startupSeq(newval)**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

**display→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**display→stopSequence(sequenceName)**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

**display→swapLayerContent(layerIdA, layerIdB)**

Permute le contenu de deux couches d'affichage.

**display→upload(pathname, content)**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

**display→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YDisplay.FindDisplay()****YDisplay****yFindDisplay()yFindDisplay( )**

Permet de retrouver un ecran d'après un identifiant donné.

**YDisplay\* yFindDisplay( string 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'écran soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDisplay.isOnLine()` pour tester si l'écran 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'écran sans ambiguïté

**Retourne :**

un objet de classe `YDisplay` qui permet ensuite de contrôler l'écran.

## **YDisplay.FirstDisplay()**

**YDisplay**

### **yFirstDisplay()yFirstDisplay( )**

Commence l'énumération des écran accessibles par la librairie.

**YDisplay\* yFirstDisplay( )**

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écran.

**Retourne :**

un pointeur sur un objet `YDisplay`, correspondant au premier écran accessible en ligne, ou `null` si il n'y a pas de écran disponibles.

**display→copyLayerContent()display→  
copyLayerContent( )****YDisplay**

Copie le contenu d'un couche d'affichage vers une autre couche.

```
int copyLayerContent( int srcLayerId, int dstLayerId)
```

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**srcLayerId** l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

**dstLayerId** l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→describe()****YDisplay**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'écran (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**display→fade()display→fade( )****YDisplay**

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

int **fade( int brightness, int duration)**

**Paramètres :**

**brightness** nouvelle valeur de luminosité de l'écran

**duration** durée en millisecondes de la transition.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→get\_advertisedValue()** YDisplay  
**display→advertisedValue()display→get\_advertisedValue( )**

---

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'écran (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**display→get\_brightness()****YDisplay****display→brightness()display→get\_brightness( )**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
int get_brightness( )
```

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne Y\_BRIGHTNESS\_INVALID.

**display→get\_displayHeight()  
display→displayHeight()display→  
get\_displayHeight( )**

---

**YDisplay**

Retourne la hauteur de l'écran, en pixels.

```
int get_displayHeight( )
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYHEIGHT\_INVALID.

**display→get\_displayLayer()**  
**display→displayLayer()display→**  
**get\_displayLayer( )**

**YDisplay**

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

**YDisplayLayer\* get\_displayLayer( unsigned layerId)**

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

**Paramètres :**

**layerId** l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

**Retourne :**

un objet YDisplayLayer

En cas d'erreur, déclenche une exception ou retourne null.

**display→get\_displayType()**  
**display→displayType()display→**  
**get\_displayType( )**

**YDisplay**

---

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

**Y\_DISPLAYTYPE\_enum get\_displayType( )**

**Retourne :**

une valeur parmi Y\_DISPLAYTYPE\_MONO, Y\_DISPLAYTYPE\_GRAY et Y\_DISPLAYTYPE\_RGB représentant le type de l'écran: monochrome, niveaux de gris ou couleur

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYTYPE\_INVALID.

---

**display→get\_displayWidth()**  
**display→displayWidth()display→**  
**get\_displayWidth( )**

---

**YDisplay**

Retourne la largeur de l'écran, en pixels.

```
int get_displayWidth( )
```

**Retourne :**

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYWIDTH\_INVALID.

**display→get\_enabled()**

**YDisplay**

**display→enabled()display→get\_enabled( )**

---

Retourne vrai si le l'écran est alimenté, faux sinon.

**Y\_ENABLED\_enum get\_enabled( )**

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon vrai si le l'écran est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

---

**display→getErrorMessage()**  
**display→errorMessage()display→**  
**getErrorMessage( )**

**YDisplay**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

**display→get\_errorType()** YDisplay  
**display→errorType()display→get\_errorType( )**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la bibliothèque Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

**display→get\_friendlyName()**  
**display→friendlyName()display→**  
**get\_friendlyName( )**

**YDisplay**

Retourne un identifiant global de l'écran au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et de l'écran si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'écran (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'écran en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

<b>display→get_functionDescriptor()</b>	<b>YDisplay</b>
<b>display→functionDescriptor()display→get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**display→get\_functionId()****YDisplay****display→functionId()display→get\_functionId( )**

---

Retourne l'identifiant matériel de l'écran, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant l'écran (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

**display→get\_hardwareId()**

**YDisplay**

**display→hardwareId()display→get\_hardwareId( )**

---

Retourne l'identifiant matériel unique de l'écran au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'écran (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'écran (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**display→get\_layerCount()****YDisplay****display→layerCount()display→get\_layerCount( )**

Retourne le nombre des couches affichables disponibles.

```
int get_layerCount( )
```

**Retourne :**

un entier représentant le nombre des couches affichables disponibles

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERCOUNT\_INVALID.

**display→get\_layerHeight()** YDisplay  
**display→layerHeight()** **display→get\_layerHeight( )**

---

Retourne la hauteur des couches affichables, en pixels.

```
int get_layerHeight( )
```

**Retourne :**

un entier représentant la hauteur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERHEIGHT\_INVALID.

**display→get\_layerWidth()****YDisplay****display→layerWidth()display→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

```
int get_layerWidth( )
```

**Retourne :**

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERWIDTH\_INVALID.

**display→get\_logicalName()** YDisplay  
**display→logicalName()** **display→get\_logicalName( )**

---

Retourne le nom logique de l'écran.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'écran. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**display→get\_module()****YDisplay****display→module()display→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**une instance de `YModule`

**display→get\_orientation()** **YDisplay**  
**display→orientation()display→**  
**get\_orientation( )**

Retourne l'orientation sélectionnée pour l'écran.

**Y\_ORIENTATION\_enum get\_orientation( )**

**Retourne :**

une valeur parmi Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP, Y\_ORIENTATION\_RIGHT et  
Y\_ORIENTATION\_DOWN représentant l'orientation sélectionnée pour l'écran

En cas d'erreur, déclenche une exception ou retourne Y\_ORIENTATION\_INVALID.

**display→get\_startupSeq()****YDisplay****display→startupSeq()display→get\_startupSeq( )**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

```
string get_startupSeq( )
```

**Retourne :**

une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

En cas d'erreur, déclenche une exception ou retourne Y\_STARTUPSEQ\_INVALID.

**display→get(userData)**

**YDisplay**

**display→userData()display→get(userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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.

**display→isOnline()****YDisplay**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de l'écran sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'écran est joignable, false sinon

**display→load()****YDisplay**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→newSequence()****YDisplay**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

```
int newSequence( )
```

Le nom de la séquence sera donné au moment de l'appel à `saveSequence( )`, une fois la séquence terminée.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **display→nextDisplay()display→nextDisplay()**

**YDisplay**

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

`YDisplay * nextDisplay()`

**Retourne :**

un pointeur sur un objet `YDisplay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**display→pauseSequence()display→  
pauseSequence( )****YDisplay**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

**int pauseSequence( int delay\_ms)**

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

**Paramètres :****delay\_ms** la durée de l'attente, en millisecondes**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→playSequence()display→  
playSequence( )**

---

**YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

**int playSequence( string sequenceName)**

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display->registerValueCallback()display->  
registerValueCallback( )****YDisplay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YDisplayValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## display→resetAll()display→resetAll( )

YDisplay

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

```
int resetAll( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→saveSequence()display→  
saveSequence( )****YDisplay**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

```
int saveSequence( string sequenceName)
```

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_brightness()** **YDisplay**  
**display→setBrightness()display→**  
**set\_brightness( )**

Modifie la luminositéde l'écran.

**int set\_brightness( int 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 luminositéde l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**display→set\_enabled()** **YDisplay**  
**display→setEnabled()display→set\_enabled( )**

Modifie l'état d'activité de l'écran.

```
int set_enabled( Y_ENABLED_enum newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état d'activité de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_logicalName()** YDisplay  
**display→setLogicalName()** **display→set\_logicalName( )**

---

Modifie le nom logique de l'écran.

```
int set_logicalName( const string& 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'écran.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_orientation()  
display→setOrientation()display→  
set\_orientation( )****YDisplay**

Modifie l'orientation de l'écran.

```
int set_orientation( Y_ORIENTATION_enum newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP, Y\_ORIENTATION\_RIGHT et Y\_ORIENTATION\_DOWN représentant l'orientation de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_startupSeq()** YDisplay  
**display→setStartupSeq()display→**  
**set\_startupSeq( )**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
int set_startupSeq( const string& newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set(userData)****YDisplay****display→setUserData()display→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**display→stopSequence()display→  
stopSequence( )**

---

**YDisplay**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

**int stopSequence( )**

L'affichage est laissé tel quel.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>display→swapLayerContent()display→</b>	<b>YDisplay</b>
<b>swapLayerContent( )</b>	

---

Permute le contenu de deux couches d'affichage.

```
int swapLayerContent( int layerIdA, int layerIdB)
```

La couleur et la transparence de tous les pixels des deux couches sont permutées. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**layerIdA** l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

**layerIdB** l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→upload()****YDisplay**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

```
int upload( string pathname, string content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.14. Interface des objets DisplayLayer

Un DisplayLayer est une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_display.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDisplay = yoctolib.YDisplay;
php require_once('yocto_display.php');
cpp #include "yocto_display.h"
m #import "yocto_display.h"
pas uses yocto_display;
vb yocto_display.vb
cs yocto_display.cs
java import com.yoctopuce.YoctoAPI.YDisplay;
py from yocto_display import *

```

### Méthodes des objets YDisplayLayer

#### **displaylayer→clear()**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

#### **displaylayer→clearConsole(text)**

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

#### **displaylayer→consoleOut(text)**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

#### **displaylayer→drawBar(x1, y1, x2, y2)**

Dessine un rectangle plein à une position spécifiée.

#### **displaylayer→drawBitmap(x, y, w, bitmap, bgcol)**

Dessine un bitmap à la position spécifiée de la couche.

#### **displaylayer→drawCircle(x, y, r)**

Dessine un cercle vide à une position spécifiée.

#### **displaylayer→drawDisc(x, y, r)**

Dessine un disque plein à une position spécifiée.

#### **displaylayer→drawImage(x, y, imagename)**

Dessine une image GIF à la position spécifiée de la couche.

#### **displaylayer→drawPixel(x, y)**

Dessine un pixel unique à une position spécifiée.

#### **displaylayer→drawRect(x1, y1, x2, y2)**

Dessine un rectangle vide à une position spécifiée.

#### **displaylayer→drawText(x, y, anchor, text)**

Affiche un texte à la position spécifiée de la couche.

#### **displaylayer→get\_display()**

Retourne l'YDisplay parent.

#### **displaylayer→get\_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

#### **displaylayer→get\_displayWidth()**

Retourne la largeur de l'écran, en pixels.

### 3. Reference

#### **displaylayer→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

#### **displaylayer→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

#### **displaylayer→hide()**

Cache la couche de dessin.

#### **displaylayer→lineTo(x, y)**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

#### **displaylayer→moveTo(x, y)**

Déplace le point de dessin courant de cette couche à la position spécifiée.

#### **displaylayer→reset()**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

#### **displaylayer→selectColorPen(color)**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

#### **displaylayer→selectEraser()**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

#### **displaylayer→selectFont(fontname)**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

#### **displaylayer→selectGrayPen(graylevel)**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

#### **displaylayer→setAntialiasingMode(mode)**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

#### **displaylayer→setConsoleBackground(bgcol)**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

#### **displaylayer→setConsoleMargins(x1, y1, x2, y2)**

Configure les marges d'affichage pour la fonction `consoleOut`.

#### **displaylayer→setConsoleWordWrap(wordwrap)**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

#### **displaylayer→setLayerPosition(x, y, scrollTime)**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

#### **displaylayer→unhide()**

Affiche la couche.

**displaylayer→clear()****YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

**int clear( )**

Cette méthode ne change pas les réglages de la couche. Si vous désirez remettre la couche dans son état initial, utilisez plutôt la méthode `reset()`.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→clearConsole()displaylayer→  
clearConsole( )**

**YDisplayLayer**

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

**int clearConsole( )**

**Paramètres :**

**text** le texte à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→consoleOut()displaylayer→  
consoleOut( )****YDisplayLayer**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

```
int consoleOut( string text)
```

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

**Paramètres :**

**text** le message à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBar()****YDisplayLayer**

Dessine un rectangle plein à une position spécifiée.

```
int drawBar( int x1, int y1, int x2, int y2)
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

**y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

**x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

**y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBitmap()displaylayer→  
drawBitmap( )****YDisplayLayer**

Dessine un bitmap à la position spécifiée de la couche.

```
int drawBitmap( int x, int y, int w, string bitmap, int bgcol)
```

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour lasser les pixels inchangés

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawCircle()** **displaylayer→drawCircle()**

---

**YDisplayLayer**

Dessine un cercle vide à une position spécifiée.

```
int drawCircle( int x, int y, int r)
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle

**y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle

**r** le rayon du cercle, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawDisc()**  
**displaylayer→  
drawDisc( )**

**YDisplayLayer**

Dessine un disque plein à une position spécifiée.

int **drawDisc( int x, int y, int r)**

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque

**y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque

**r** le rayon du disque, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawImage()displaylayer→  
drawImage( )**

**YDisplayLayer**

Dessine une image GIF à la position spécifiée de la couche.

**int drawImage( int x, int y, string imagename)**

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image

**y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image

**imagename** le nom du fichier GIF à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawPixel()  
displaylayer→  
drawPixel( )****YDisplayLayer**

Dessine un pixel unique à une position spécifiée.

```
int drawPixel( int x, int y)
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche

**y** la distance en pixels depuis le haut de la couche

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawRect()** **displaylayer→drawRect( )**

---

**YDisplayLayer**

Dessine un rectangle vide à une position spécifiée.

int **drawRect( int x1, int y1, int x2, int y2)**

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

**y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

**x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

**y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawText()displaylayer→  
drawText( )****YDisplayLayer**

Affiche un texte à la position spécifiée de la couche.

```
int drawText( int x, int y, Y_ALIGN anchor, string text)
```

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte  
**y** la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte  
**anchor** le point d'ancrage du texte, choisi parmi l'énumération Y\_ALIGN: Y\_ALIGN\_TOP\_LEFT,  
Y\_ALIGN\_CENTER\_LEFT, Y\_ALIGN\_BASELINE\_LEFT, Y\_ALIGN\_BOTTOM\_LEFT,  
Y\_ALIGN\_TOP\_CENTER, Y\_ALIGN\_CENTER, Y\_ALIGN\_BASELINE\_CENTER,  
Y\_ALIGN\_BOTTOM\_CENTER, Y\_ALIGN\_TOP\_DECIMAL,  
Y\_ALIGN\_CENTER\_DECIMAL, Y\_ALIGN\_BASELINE\_DECIMAL,  
Y\_ALIGN\_BOTTOM\_DECIMAL, Y\_ALIGN\_TOP\_RIGHT, Y\_ALIGN\_CENTER\_RIGHT,  
Y\_ALIGN\_BASELINE\_RIGHT, Y\_ALIGN\_BOTTOM\_RIGHT.  
**text** le texte à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→get\_display()**  
**displaylayer→display()displaylayer→**  
**get\_display( )**

---

**YDisplayLayer**

Retourne l'YDisplay parent.

**YDisplay\* get\_display( )**

Retourne l'objet YDisplay parent du YDisplayLayer courant.

**Retourne :**

un objet YDisplay

**displaylayer→get\_displayHeight()**  
**displaylayer→displayHeight()displaylayer→**  
**get\_displayHeight( )**

**YDisplayLayer**

Retourne la hauteur de l'écran, en pixels.

```
int get_displayHeight( )
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYHEIGHT\_INVALID.

**displaylayer→get\_displayWidth()**  
**displaylayer→displayWidth()displaylayer→**  
**get\_displayWidth( )**

---

**YDisplayLayer**

Retourne la largeur de l'écran, en pixels.

**int get\_displayWidth( )**

**Retourne :**

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYWIDTH\_INVALID.

**displaylayer→get\_layerHeight()**  
**displaylayer→layerHeight()** **displaylayer→get\_layerHeight( )**

**YDisplayLayer**

Retourne la hauteur des couches affichables, en pixels.

```
int get_layerHeight( )
```

**Retourne :**

un entier représentant la hauteur des couches affichables, en pixels. En cas d'erreur, déclenche une exception ou retourne Y\_LAYERHEIGHT\_INVALID.

**displaylayer→get\_layerWidth()**  
**displaylayer→layerWidth()displaylayer→**  
**get\_layerWidth( )**

**YDisplayLayer**

---

Retourne la largeur des couches affichables, en pixels.

**int get\_layerWidth( )**

**Retourne :**

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERWIDTH\_INVALID.

**displaylayer→hide()****YDisplayLayer**

Cache la couche de dessin.

**int hide( )**

L'état de la couche est préservé, mais la couche ne sera plus affichée à l'écran jusqu'au prochain appel à `unhide()`. Le fait de cacher la couche améliore les performances de toutes les primitives d'affichage, car il évite de consacrer inutilement des cycles de calcul à afficher les états intermédiaires (technique de double-buffering).

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→lineTo()displaylayer→lineTo( )****YDisplayLayer**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

```
int lineTo( int x, int y)
```

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au point final

**y** la distance en pixels depuis le haut de la couche jusqu'au point final

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→moveTo()displaylayer→moveTo( )****YDisplayLayer**

Déplace le point de dessin courant de cette couche à la position spécifiée.

```
int moveTo( int x, int y)
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche de dessin

**y** la distance en pixels depuis le haut de la couche de dessin

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→reset()displaylayer→reset()****YDisplayLayer**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

```
int reset( )
```

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear()`.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectColorPen()displaylayer→  
selectColorPen( )**

**YDisplayLayer**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

**int selectColorPen( int color)**

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

**Paramètres :**

**color** la couleur RGB désirée (sous forme d'entier 24 bits)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectEraser()displaylayer→  
selectEraser( )**

**YDisplayLayer**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

**int selectEraser( )**

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectFont()displaylayer→  
selectFont( )****YDisplayLayer**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

**int selectFont( string fontname)**

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

**Paramètres :**

**fontname** le nom du fichier définissant la police de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectGrayPen()displaylayer→  
selectGrayPen( )**

**YDisplayLayer**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

**int selectGrayPen( int graylevel)**

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), tout valeur inférieure à 128 conduit à un point noir, et toute valeur supérieure ou égale à 128 devient un point lumineux.

**Paramètres :**

**graylevel** le niveau de gris désiré, de 0 à 255

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setAntialiasingMode()**  
**displaylayer→setAntialiasingMode( )****YDisplayLayer**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

```
int setAntialiasingMode( bool mode)
```

L'anti-aliasing est atténué la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

**Paramètres :**

**mode** true pour activer l'antialiasing, false pour le désactiver.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleBackground()****YDisplayLayer****displaylayer→setConsoleBackground( )**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

```
int setConsoleBackground( int bgcol)
```

**Paramètres :**

**bgcol** le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleMargins()** **displaylayer→setConsoleMargins( )**

**YDisplayLayer**

Configure les marges d'affichage pour la fonction `consoleOut`.

```
int setConsoleMargins( int x1, int y1, int x2, int y2)
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche

**y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure

**x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite

**y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleWordWrap()displaylayer  
→setConsoleWordWrap( )**

---

**YDisplayLayer**

Configure le mode de retour à la ligne utilisé par la fonction consoleOut.

```
int setConsoleWordWrap( bool wordwrap)
```

**Paramètres :**

**wordwrap** true pour retourner à la ligne entre les mots seulement, false pour retourner à l'extrême droite de chaque ligne.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setLayerPosition()****YDisplayLayer****setLayerPosition()**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

```
int setLayerPosition( int x, int y, int scrollTime)
```

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

**Paramètres :**

- x** la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
- y** la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
- scrollTime** durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## displaylayer→unhide() displaylayer→unhide()

YDisplayLayer

Affiche la couche.

```
int unhide( )
```

Affiche à nouveau la couche après la commande hide.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.15. Interface de contrôle de l'alimentation

La librairie de programmation Yoctopuce permet de contrôler la source d'alimentation qui doit être utilisée pour les fonctions du module consommant beaucoup de courant. Le module est par ailleurs capable de couper automatiquement l'alimentation externe lorsqu'il détecte que la tension a trop chuté (batterie épuisée).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_dualpower.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YDualPower = yoctolib.YDualPower;
php	require_once('yocto_dualpower.php');
cpp	#include "yocto_dualpower.h"
m	#import "yocto_dualpower.h"
pas	uses yocto_dualpower;
vb	yocto_dualpower.vb
cs	yocto_dualpower.cs
java	import com.yoctopuce.YoctoAPI.YDualPower;
py	from yocto_dualpower import *

### Fonction globales

#### yFindDualPower(func)

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

#### yFirstDualPower()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

### Méthodes des objets YDualPower

#### dualpower→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### dualpower→get\_advertisedValue()

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

#### dualpower→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

#### dualpower→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

#### dualpower→get\_extVoltage()

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

#### dualpower→get\_friendlyName()

Retourne un identifiant global du contrôle d'alimentation au format NOM\_MODULE . NOM\_FONCTION.

#### dualpower→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### dualpower→get\_functionId()

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

#### dualpower→get\_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'alimentation au format SERIAL . FUNCTIONID.

#### dualpower→get\_logicalName()

### 3. Reference

Retourne le nom logique du contrôle d'alimentation.

#### **dualpower→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **dualpower→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **dualpower→get\_powerControl()**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

#### **dualpower→get\_powerState()**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

#### **dualpower→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **dualpower→isOnline()**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

#### **dualpower→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

#### **dualpower→load(msValidity)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

#### **dualpower→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

#### **dualpower→nextDualPower()**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de yFirstDualPower( ).

#### **dualpower→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **dualpower→set\_logicalName(newval)**

Modifie le nom logique du contrôle d'alimentation.

#### **dualpower→set\_powerControl(newval)**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

#### **dualpower→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **dualpower→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YDualPower.FindDualPower() yFindDualPower()yFindDualPower( )

## YDualPower

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

**YDualPower\* yFindDualPower( const string& 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()  
yFirstDualPower()yFirstDualPower( )**

---

**YDualPower**

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

**YDualPower\* yFirstDualPower( )**

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

**Retourne :**

un pointeur sur un objet `YDualPower`, correspondant au premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

**dualpower→describe()****YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le contrôle d'alimentation (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**dualpower→get\_advertisedValue()**  
**dualpower→advertisedValue()****dualpower→get\_advertisedValue( )**

---

**YDualPower**

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

**string 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→getErrorMessage()**  
**dualpower→errorMessage()****dualpower→getErrorMessage( )**

**YDualPower**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

**dualpower→get\_errorType()**  
**dualpower→errorType()**  
**dualpower→get\_errorType( )**

---

**YDualPower**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

**dualpower→get\_extVoltage()**  
**dualpower→extVoltage()****dualpower→get\_extVoltage( )**

**YDualPower**

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

**int get\_extVoltage( )**

**Retourne :**

un entier représentant la tension mesurée sur l'alimentation de puissance externe, en millivolts

En cas d'erreur, déclenche une exception ou retourne Y\_EXTVOLTAGE\_INVALID.

**dualpower→get\_friendlyName()**  
**dualpower→friendlyName()****dualpower→get\_friendlyName( )**

**YDualPower**

Retourne un identifiant global du contrôle d'alimentation au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du contrôle d'alimentation si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'alimentation (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**dualpower→get\_functionDescriptor()**  
**dualpower→functionDescriptor()dualpower→**  
**get\_functionDescriptor( )**

**YDualPower**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**dualpower→get\_functionId()**  
**dualpower→functionId()****dualpower→get\_functionId( )**

---

**YDualPower**

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

```
string get_functionId( )
```

Par example `relay1`.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**dualpower→get\_hardwareId()**

**YDualPower**

**dualpower→hardwareId()** **dualpower→get\_hardwareId( )**

Retourne l'identifiant matériel unique du contrôle d'alimentation au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'alimentation (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**dualpower→get\_logicalName()**  
**dualpower→logicalName()****dualpower→get\_logicalName( )**

---

**YDualPower**

Retourne le nom logique du contrôle d'alimentation.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du contrôle d'alimentation. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**dualpower→get\_module()****YDualPower****dualpower→module()dualpower→get\_module( )**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`YModule * get_module( )`**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**une instance de `YModule`

**dualpower→get\_powerControl()**  
**dualpower→powerControl()****dualpower→get\_powerControl( )**

---

**YDualPower**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

**Y\_POWERCONTROL\_enum get\_powerControl( )**

**Retourne :**

une valeur parmi Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT et Y\_POWERCONTROL\_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y\_POWERCONTROL\_INVALID.

**dualpower→get\_powerState()** YDualPower  
**dualpower→powerState()** **dualpower→get\_powerState( )**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

**Y\_POWERSTATE\_enum 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())**  
**dualpower→userData()****dualpower→**  
**get(userData())**

---

**YDualPower**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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()****YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le contrôle d'alimentation est joignable, false sinon

**dualpower→load()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→nextDualPower()**  
**dualpower→**  
**nextDualPower( )**

**YDualPower**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de **yFirstDualPower( ).**

**YDualPower \* nextDualPower( )**

**Retourne :**

un pointeur sur un objet **YDualPower** accessible en ligne, ou **null** lorsque l'énumération est terminée.

**dualpower→registerValueCallback()**  
**dualpower→registerValueCallback( )**

**YDualPower**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**int registerValueCallback( YDualPowerValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**dualpower→set\_logicalName()**  
**dualpower→setLogicalName()****dualpower→**  
**set\_logicalName( )**

**YDualPower**

Modifie le nom logique du contrôle d'alimentation.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser **yCheckLogicalName( )** pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode **saveToFlash( )** du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

**Retourne :**

**YAPI\_SUCCESS** si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→set\_powerControl()** **YDualPower**  
**dualpower→setPowerControl()** **dualpower→**  
**set\_powerControl( )**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
int set_powerControl( Y_POWERCONTROL_enum 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())**  
**dualpower→setUserData()****dualpower→**  
**set(userData())**

**YDualPower**

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).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.16. Interface de la fonction Files

L'interface de stockage de fichiers permet de stocker des fichiers sur certains modules, par exemple pour personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_files.js'></script>
nodejs var yoctolib = require('yoctolib');
var YFiles = yoctolib.YFiles;
php require_once('yocto_files.php');
cpp #include "yocto_files.h"
m #import "yocto_files.h"
pas uses yocto_files;
vb yocto_files.vb
cs yocto_files.cs
java import com.yoctopuce.YoctoAPI.YFiles;
py from yocto_files import *

```

### Fonction globales

#### **yFindFiles(func)**

Permet de retrouver un système de fichier d'après un identifiant donné.

#### **yFirstFiles()**

Commence l'énumération des système de fichier accessibles par la librairie.

### Méthodes des objets YFiles

#### **files→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **files→download(pathname)**

Télécharge le fichier choisi du filesystème et retourne son contenu.

#### **files→download\_async(pathname, callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

#### **files→format\_fs()**

Rétablissement le système de fichier dans un état original, défragmenté.

#### **files→get\_advertisedValue()**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

#### **files→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### **files→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### **files→get\_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

#### **files→get\_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

#### **files→get\_friendlyName()**

Retourne un identifiant global du système de fichier au format NOM\_MODULE . NOM\_FONCTION.

#### **files→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**files→get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

**files→get\_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

**files→get\_list(pattern)**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

**files→get\_logicalName()**

Retourne le nom logique du système de fichier.

**files→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**files→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**files→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**files→isOnline()**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**files→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**files→load(msValidity)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**files→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**files→nextFiles()**

Continue l'énumération des système de fichier commencée à l'aide de yFirstFiles( ).

**files→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**files→remove(pathname)**

Efface un fichier, spécifié par son path complet, du système de fichier.

**files→set\_logicalName(newval)**

Modifie le nom logique du système de fichier.

**files→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**files→upload(pathname, content)**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

**files→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YFiles.FindFiles()****YFiles****yFindFiles()yFindFiles( )**

Permet de retrouver un système de fichier d'après un identifiant donné.

YFiles\* **yFindFiles( string func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le système de fichier sans ambiguïté

**Retourne :**

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

**YFiles.FirstFiles()****YFiles****yFirstFiles()yFirstFiles( )**

Commence l'énumération des système de fichier accessibles par la librairie.

**YFiles\* yFirstFiles( )**

Utiliser la fonction `YFiles.nextFiles( )` pour itérer sur les autres système de fichier.

**Retourne :**

un pointeur sur un objet `YFiles`, correspondant au premier système de fichier accessible en ligne, ou null si il n'y a pas de système de fichier disponibles.

**files→describe()****YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE (NAME )=SERIAL.FUNCTIONID.

string **describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le système de fichier (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**files→download()****YFiles**

Télécharge le fichier choisi du système et retourne son contenu.

```
string download( string pathname)
```

**Paramètres :**

**pathname** nom complet du fichier à charger, y compris le chemin d'accès.

**Retourne :**

le contenu du fichier chargé sous forme d'objet binaire

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

## files→format\_fs()files→format\_fs()

YFiles

Rétabli le système de fichier dans un état original, défragmenté.

`int format_fs( )`

entièrement vide. Tous les fichiers précédemment chargés sont irrémédiablement effacés.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→get\_advertisedValue()**  
**files→advertisedValue()files→**  
**get\_advertisedValue( )**

**YFiles**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**YFiles**  
**files→getErrorMessage()**  
**files→errorMessage()files→getErrorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

**files→get\_errorType()****YFiles****files→errorType()files→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

**files→get\_filesCount()** YFiles  
**files→filesCount()files→get\_filesCount( )**

---

Retourne le nombre de fichiers présents dans le système de fichier.

**int get\_filesCount( )**

**Retourne :**

un entier représentant le nombre de fichiers présents dans le système de fichier

En cas d'erreur, déclenche une exception ou retourne Y\_FILESCOUNT\_INVALID.

**files→get\_freeSpace()****YFiles****files→freeSpace()files→get\_freeSpace( )**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

```
int get_freeSpace( )
```

**Retourne :**

un entier représentant l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets

En cas d'erreur, déclenche une exception ou retourne Y\_FREESPACE\_INVALID.

---

<b>files→get_friendlyName()</b>	<b>YFiles</b>
<b>files→friendlyName()files→get_friendlyName( )</b>	

---

Retourne un identifiant global du système de fichier au format NOM\_MODULE.NOM\_FONCTION.

```
string get_friendlyName( )
```

Le chaîne renvoyée utilise soit les noms logiques du module et du système de fichier si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du système de fichier (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le système de fichier en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**files→get\_functionDescriptor()**  
**files→functionDescriptor()files→**  
**get\_functionDescriptor( )**

**YFiles**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>files→get_functionId()</b>	<b>YFiles</b>
<b>files→functionId()files→get_functionId()</b>	

---

Retourne l'identifiant matériel du système de fichier, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**files→get\_hardwareId()****YFiles****files→hardwareId()files→get\_hardwareId( )**

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**files→get\_list()****YFiles****files→list()files→get\_list( )**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
vector<YFileRecord> get_list( string pattern)
```

**Paramètres :**

**pattern** un filtre optionnel sur les noms de fichiers retournés, pouvant contenir des astérisques et des points d'interrogations comme jokers. Si le pattern fourni est vide, tous les fichiers sont retournés.

**Retourne :**

une liste d'objets YFileRecord, contenant le nom complet (y compris le chemin d'accès), la taille en octets et le CRC 32-bit du contenu du fichier.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**files→get\_logicalName()****YFiles****files→logicalName()files→get\_logicalName( )**

Retourne le nom logique du système de fichier.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du système de fichier. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

<b>files→get_module()</b>	<b>YFiles</b>
<b>files→module()files→get_module( )</b>	

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule * get_module( )`

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**files→get(userData)****YFiles****files→userData()files→get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**files→isOnline()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le système de fichier est joignable, false sinon

**files→load()****YFiles**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files->nextFiles()files->nextFiles( )**

**YFiles**

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles( )`.

`YFiles * nextFiles( )`

**Retourne :**

un pointeur sur un objet `YFiles` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**files→registerValueCallback()**  
**files→registerValueCallback( )****YFiles**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YFilesValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**files→remove()****YFiles**

Efface un fichier, spécifié par son path complet, du système de fichier.

**int remove( string pathname)**

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occupe. Par contre, la ré-écriture d'un fichier du même nom récupérera dans tout les cas l'espace qui n'aurait éventuellement pas été libéré. Pour s'assurer de libérer la totalité de l'espace du système de fichier, utilisez la fonction `format_fs`.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→set\_logicalName()**  
**files→setLogicalName()****files→**  
**set\_logicalName( )**

**YFiles**

Modifie le nom logique du système de fichier.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du système de fichier.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**YFiles**  
**files→set(userData)**  
**files→setUserData()files→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**files→upload()****YFiles**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

```
int upload( string pathname, string content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.17. Interface de la fonction GenericSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_genericsensor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGenericSensor = yoctolib.YGenericSensor;
require_once('yocto_genericsensor.php');
#include "yocto_genericsensor.h"
m #import "yocto_genericsensor.h"
pas uses yocto_genericsensor;
vb yocto_genericsensor.vb
cs yocto_genericsensor.cs
java import com.yoctopuce.YoctoAPI.YGenericSensor;
py from yocto_genericsensor import *

```

### Fonction globales

#### yFindGenericSensor(func)

Permet de retrouver un capteur générique d'après un identifiant donné.

#### yFirstGenericSensor()

Commence l'énumération des capteurs génériques accessibles par la librairie.

### Méthodes des objets YGenericSensor

#### genericsensor→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### genericsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### genericsensor→get\_advertisedValue()

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

#### genericsensor→get\_currentRawValue()

Retourne la valeur brute rentrée par le capteur (sans arrondi ni calibration).

#### genericsensor→get\_currentValue()

Retourne la valeur mesurée actuelle.

#### genericsensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### genericsensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### genericsensor→get\_friendlyName()

Retourne un identifiant global du capteur générique au format NOM\_MODULE . NOM\_FONCTION.

#### genericsensor→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### genericsensor→get\_functionId()

Retourne l'identifiant matériel du capteur générique, sans référence au module.

#### genericsensor→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur générique au format SERIAL . FUNCTIONID.

#### **genericsensor→get\_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

#### **genericsensor→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

#### **genericsensor→get\_logicalName()**

Retourne le nom logique du capteur générique.

#### **genericsensor→get\_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

#### **genericsensor→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **genericsensor→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **genericsensor→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

#### **genericsensor→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

#### **genericsensor→get\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **genericsensor→get\_signalRange()**

Retourne la plage de signal électrique utilisée par le capteur.

#### **genericsensor→get\_signalUnit()**

Retourne l'unité du signal électrique utilisée par le capteur.

#### **genericsensor→get\_signalValue()**

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

#### **genericsensor→get\_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

#### **genericsensor→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **genericsensor→get\_valueRange()**

Retourne la plage de valeurs physiques mesurés par le capteur.

#### **genericsensor→isOnline()**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

#### **genericsensor→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

#### **genericsensor→load(msValidity)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

#### **genericsensor→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **genericsensor→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

#### **genericsensor→nextGenericSensor()**

### 3. Reference

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

#### `genericSensor->registerTimedReportCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### `genericSensor->registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### `genericSensor->set_highestValue(newval)`

Modifie la mémoire de valeur maximale observée.

#### `genericSensor->set_logFrequency(newval)`

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### `genericSensor->set_logicalName(newval)`

Modifie le nom logique du capteur générique.

#### `genericSensor->set_lowestValue(newval)`

Modifie la mémoire de valeur minimale observée.

#### `genericSensor->set_reportFrequency(newval)`

Modifie la fréquence de notification périodique des valeurs mesurées.

#### `genericSensor->set_resolution(newval)`

Modifie la résolution des valeurs physique mesurées.

#### `genericSensor->set_signalRange(newval)`

Modifie la plage de signal électrique utilisée par le capteur.

#### `genericSensor->set_unit(newval)`

Change l'unité dans laquelle la valeur mesurée est exprimée.

#### `genericSensor->set_userData(data)`

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

#### `genericSensor->set_valueRange(newval)`

Modifie la plage de valeurs physiques mesurés par le capteur.

#### `genericSensor->wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YGenericSensor.FindGenericSensor()****yFindGenericSensor()yFindGenericSensor( )****YGenericSensor**

Permet de retrouver un capteur générique d'après un identifiant donné.

**YGenericSensor\* yFindGenericSensor( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur générique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGenericSensor.isOnLine()` pour tester si le capteur générique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur générique sans ambiguïté

**Retourne :**

un objet de classe `YGenericSensor` qui permet ensuite de contrôler le capteur générique.

## **YGenericSensor.FirstGenericSensor()**

## **YGenericSensor**

### **yFirstGenericSensor()yFirstGenericSensor( )**

Commence l'énumération des capteurs génériques accessibles par la librairie.

**YGenericSensor\* yFirstGenericSensor( )**

Utiliser la fonction `YGenericSensor.nextGenericSensor( )` pour itérer sur les autres capteurs génériques.

**Retourne :**

un pointeur sur un objet `YGenericSensor`, correspondant au premier capteur générique accessible en ligne, ou `null` si il n'y a pas de capteurs génériques disponibles.

**genericsensor→calibrateFromPoints()****YGenericSensor****genericsensor→calibrateFromPoints( )**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→describe()genericsensor→  
describe()****YGenericSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format TYPE(NAME)=SERIAL.FUNCTIONID.

```
string describe()
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur générique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

---

<b>genericsensor→get_advertisedValue()</b>	<b>YGenericSensor</b>
<b>genericsensor→advertisedValue()genericsensor→get_advertisedValue( )</b>	

---

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

```
string get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur générique (pas plus de 6 caractères).  
En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**genericsensor→get\_currentRawValue()**  
**genericsensor→currentRawValue()genericsensor**  
**→get\_currentRawValue()**

---

**YGenericSensor**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

double **get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**genericsensor→get\_currentValue()**  
**genericsensor→currentValue()****genericsensor→get\_currentValue()**

**YGenericSensor**

Retourne la valeur mesurée actuelle.

**double get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**genericsensor→get\_errorMessage()**  
**genericsensor→errorMessage()genericsensor→**  
**get\_errorMessage( )**

---

**YGenericSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

---

```
genericSensor->get_errorType()  
genericSensor->errorType()genericSensor->  
get_errorType( )
```

**YGenericSensor**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

**genericsensor→get\_friendlyName()** **YGenericSensor**  
**genericsensor→friendlyName()****genericsensor→get\_friendlyName( )**

---

Retourne un identifiant global du capteur générique au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur générique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur générique (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur générique en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

`genericSensor->get_functionDescriptor()`

**YGenericSensor**

`genericSensor->functionDescriptor()genericSensor->get_functionDescriptor()`

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**genericsensor→get\_functionId()** **YGenericSensor**  
**genericsensor→functionId()** **genericsensor→get\_functionId( )**

---

Retourne l'identifiant matériel du capteur générique, sans référence au module.

**string get\_functionId( )**

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

`genericsensor→get_hardwareId()`

**YGenericSensor**

`genericsensor→hardwareId() genericsensor→`  
`get_hardwareId( )`

Retourne l'identifiant matériel unique du capteur générique au format SERIAL.FUNCTIONID.

`string get_hardwareId( )`

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur générique (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**genericsensor→get\_highestValue()**

**YGenericSensor**

**genericsensor→highestValue()genericsensor→**

**get\_highestValue( )**

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

**double get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**genericsensor→get\_logFrequency()**

**YGenericSensor**

**genericsensor→logFrequency()genericsensor→**

**get\_logFrequency( )**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**genericsensor→get\_logicalName()** **YGenericSensor**  
**genericsensor→logicalName()****genericsensor→get\_logicalName( )**

---

Retourne le nom logique du capteur générique.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur générique. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**genericsensor→get\_lowestValue()**  
**genericsensor→lowestValue()genericsensor→**  
**get\_lowestValue( )**

**YGenericSensor**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

**double get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**genericsensor→get\_module()**  
**genericsensor→module()****genericsensor→get\_module()**

---

**YGenericSensor**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**genericsensor→get\_recordedData()**

**YGenericSensor**

**genericsensor→recordedData() genericsensor→  
get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**genericsensor→get\_reportFrequency()**

**YGenericSensor**

**genericsensor→reportFrequency()genericsensor→  
get\_reportFrequency( )**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**string get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**genericsensor→get\_resolution()**  
**genericsensor→resolution()genericsensor→**  
**get\_resolution( )**

**YGenericSensor**

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**genericsensor→get\_signalRange()**

**YGenericSensor**

**genericsensor→signalRange()genericsensor→  
get\_signalRange( )**

---

Retourne la plage de signal électrique utilisée par le capteur.

**string get\_signalRange( )**

**Retourne :**

une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALRANGE\_INVALID.

**genericsensor→get\_signalUnit()**  
**genericsensor→signalUnit()genericsensor→**  
**get\_signalUnit( )**

**YGenericSensor**

Retourne l'unité du signal électrique utilisée par le capteur.

**string get\_signalUnit( )**

**Retourne :**

une chaîne de caractères représentant l'unité du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALUNIT\_INVALID.

`genericsensor→get_signalValue()`

**YGenericSensor**

`genericsensor→signalValue()genericsensor→`

`get_signalValue()`

---

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

`double get_signalValue( )`

**Retourne :**

une valeur numérique représentant la valeur mesurée du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALVALUE_INVALID`.

---

**genericsensor→get\_unit()****YGenericSensor****genericsensor→unit()genericsensor→get\_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**genericsensor→get(userData)**

**YGenericSensor**

**genericsensor→userData()**  
**genericsensor→get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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.

**genericsensor→get\_valueRange()**  
**genericsensor→valueRange()****genericsensor→**  
**get\_valueRange( )**

**YGenericSensor**

Retourne la plage de valeurs physiques mesurés par le capteur.

**string get\_valueRange( )**

**Retourne :**

une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_VALUERANGE\_INVALID.

**genericsensor**→**isOnline()****genericsensor**→  
**isOnline( )**

---

**YGenericSensor**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du capteur générique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur générique est joignable, false sinon

**genericsensor→load()****YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→loadCalibrationPoints()****YGenericSensor****genericsensor→loadCalibrationPoints( )**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→nextGenericSensor()**

**YGenericSensor**

**genericsensor→nextGenericSensor( )**

Continue l'énumération des capteurs génériques commencée à l'aide de **yFirstGenericSensor()**.

**YGenericSensor \* nextGenericSensor( )**

**Retourne :**

un pointeur sur un objet **YGenericSensor** accessible en ligne, ou null lorsque l'énumération est terminée.

**genericsensor→registerTimedReportCallback()**

**YGenericSensor**

**genericsensor→**

**registerTimedReportCallback( )**

---

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**int registerTimedReportCallback( YGenericSensorTimedReportCallback **callback**)**

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**genericsensor→registerValueCallback()**  
**genericsensor→registerValueCallback( )**

**YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

int **registerValueCallback( YGenericSensorValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**genericsensor→set\_highestValue()** **YGenericSensor**  
**genericsensor→setHighestValue()**  
**genericsensor→set\_highestValue( )**

---

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`genericsensor→set_logFrequency()`

**YGenericSensor**

`genericsensor→setLogFrequency()genericsensor  
→set_logFrequency( )`

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

`int set_logFrequency( const string& newval)`

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

`newval` une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_logicalName()** **YGenericSensor**  
**genericsensor→setLogicalName()****genericsensor→set\_logicalName( )**

Modifie le nom logique du capteur générique.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur générique.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_lowestValue()**

**YGenericSensor**

**genericsensor→setLowestValue()** **genericsensor→**  
**set\_lowestValue( )**

Modifie la mémoire de valeur minimale observée.

**int set\_lowestValue( double newval)**

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_reportFrequency()****YGenericSensor****genericsensor→setReportFrequency()****genericsensor→set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_resolution()**

**YGenericSensor**

**genericsensor→setResolution()genericsensor→  
set\_resolution( )**

---

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_signalRange()** **YGenericSensor**  
**genericsensor→setSignalRange()****genericsensor→**  
**set\_signalRange( )**

---

Modifie la plage de signal électrique utilisée par le capteur.

```
int set_signalRange( const string& newval)
```

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>genericsensor→set_unit()</b>	<b>YGenericSensor</b>
<b>genericsensor→setUnit()genericsensor→</b> <b>set_unit( )</b>	

---

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
int set_unit( const string& newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set(userData)**  
**genericsensor→setUserData()****genericsensor→set(userData)**

**YGenericSensor**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**void set(userData void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**genericsensor→set\_valueRange()**  
**genericsensor→setValueRange()** genericSensor→  
**set\_valueRange( )**

**YGenericSensor**

Modifie la plage de valeurs physiques mesurés par le capteur.

**int set\_valueRange( const string& newval)**

Le changement de plage peut avoir pour effet de bord un changement automatique de la résolution affichée.

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.18. Interface de la fonction Gyro

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_gyro.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGyro = yoctolib.YGyro;
php require_once('yocto_gyro.php');
cpp #include "yocto_gyro.h"
m #import "yocto_gyro.h"
pas uses yocto_gyro;
vb yocto_gyro.vb
cs yocto_gyro.cs
java import com.yoctopuce.YoctoAPI.YGyro;
py from yocto_gyro import *

```

### Fonction globales

#### **yFindGyro(func)**

Permet de retrouver un gyroscope d'après un identifiant donné.

#### **yFirstGyro()**

Commence l'énumération des gyroscopes accessibles par la librairie.

### Méthodes des objets YGyro

#### **gyro→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **gyro→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format TYPE (NAME) = SERIAL . FUNCTIONID.

#### **gyro→get\_advertisedValue()**

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

#### **gyro→get\_currentRawValue()**

Retourne la valeur brute renournée par le capteur (sans arrondi ni calibration).

#### **gyro→get\_currentValue()**

Retourne la valeur actuelle de la vitesse angulaire.

#### **gyro→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

#### **gyro→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

#### **gyro→get\_friendlyName()**

Retourne un identifiant global du gyroscope au format NOM\_MODULE . NOM\_FONCTION.

#### **gyro→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **gyro→get\_functionId()**

Retourne l'identifiant matériel du gyroscope, sans référence au module.

#### **gyro→get\_hardwareId()**

Retourne l'identifiant matériel unique du gyroscope au format SERIAL . FUNCTIONID.

**gyro→get\_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

**gyro→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**gyro→get\_logicalName()**

Retourne le nom logique du gyroscope.

**gyro→get\_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

**gyro→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**gyro→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**gyro→get\_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionX()**

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionY()**

Retourne la composante y du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionZ()**

Retourne la composante z du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**gyro→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**gyro→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**gyro→get\_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

**gyro→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**gyro→get\_xValue()**

	Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.
<b>gyro→get_yValue()</b>	Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.
<b>gyro→get_zValue()</b>	Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.
<b>gyro→isOnline()</b>	Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.
<b>gyro→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.
<b>gyro→load(msValidity)</b>	Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.
<b>gyro→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>gyro→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.
<b>gyro→nextGyro()</b>	Continue l'énumération des gyroscopes commencée à l'aide de yFirstGyro( ).
<b>gyro→registerAnglesCallback(callback)</b>	Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.
<b>gyro→registerQuaternionCallback(callback)</b>	Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.
<b>gyro→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>gyro→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>gyro→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.
<b>gyro→set_logFrequency(newval)</b>	Modifie la fréquence d'enregistrement des mesures dans le datalogger.
<b>gyro→set_logicalName(newval)</b>	Modifie le nom logique du gyroscope.
<b>gyro→set_lowestValue(newval)</b>	Modifie la mémoire de valeur minimale observée.
<b>gyro→set_reportFrequency(newval)</b>	Modifie la fréquence de notification périodique des valeurs mesurées.
<b>gyro→set_resolution(newval)</b>	Modifie la résolution des valeurs physique mesurées.
<b>gyro→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>gyro→wait_async(callback, context)</b>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YGyro.FindGyro()****YGyro****yFindGyro()yFindGyro( )**

Permet de retrouver un gyroscope d'après un identifiant donné.

**YGyro\* yFindGyro( string func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le gyroscope sans ambiguïté

**Retourne :**

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

## YGyro.FirstGyro()

YGyro

### yFirstGyro()yFirstGyro( )

Commence l'énumération des gyroscopes accessibles par la librairie.

YGyro\* **yFirstGyro( )**

Utiliser la fonction YGyro.nextGyro( ) pour itérer sur les autres gyroscopes.

**Retourne :**

un pointeur sur un objet YGyro, correspondant au premier gyroscope accessible en ligne, ou null si il n'y a pas de gyroscopes disponibles.

**gyro→calibrateFromPoints()gyro→  
calibrateFromPoints( )****YGyro**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→describe()****YGyro**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le gyroscope (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

---

<b>gyro→get_advertisedValue()</b>	<b>YGyro</b>
<b>gyro→advertisedValue()</b>	<b>YGyro→get_advertisedValue( )</b>

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

```
string get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du gyroscope (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**gyro→get\_currentRawValue()**  
**gyro→currentRawValue()** gyro→  
**get\_currentRawValue( )**

---

**YGyro**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

double **get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

---

**gyro→get\_currentValue()****YGyro****gyro→currentValue()gyro→get\_currentValue()**

Retourne la valeur actuelle de la vitesse angulaire.

```
double get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la vitesse angulaire

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**gyro→getErrorMessage()**

**YGyro**

**gyro→errorMessage()gyro→getErrorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

**gyro→get\_errorType()****YGyro****gyro→errorType()gyro→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

**gyro→get\_friendlyName()****YGyro****gyro→friendlyName()gyro→get\_friendlyName( )**

Retourne un identifiant global du gyroscope au format NOM\_MODULE.NOM\_FONCTION.

```
string get_friendlyName( )
```

Le chaîne renvoyée utilise soit les noms logiques du module et du gyroscope si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du gyroscope (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le gyroscope en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**gyro→get\_functionDescriptor()**  
**gyro→functionDescriptor()** gyro→  
**get\_functionDescriptor( )**

**YGyro**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>gyro→get_functionId()</b>	<b>YGyro</b>
<b>gyro→functionId()gyro→get_functionId( )</b>	

---

Retourne l'identifiant matériel du gyroscope, sans référence au module.

string **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**gyro→get\_hardwareId()****YGyro****gyro→hardwareId()gyro→get\_hardwareId( )**

Retourne l'identifiant matériel unique du gyroscope au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

<b>gyro→get_heading()</b>	<b>YGyro</b>
<b>gyro→heading()</b>	<b>gyro→get_heading( )</b>

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**double get\_heading( )**

L'axe de lacet peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe YRefFrame.

**Retourne :**

un nombre à virgule correspondant au cap, exprimé en degrés (entre 0 et 360).

**gyro→get\_highestValue()****YGyro****gyro→highestValue()gyro→get\_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

```
double get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**gyro→get\_logFrequency()** YGyro  
**gyro→logFrequency()** **gyro→get\_logFrequency( )**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
string get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

**gyro→get\_logicalName()****YGyro****gyro→logicalName()gyro→get\_logicalName( )**

---

Retourne le nom logique du gyroscope.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du gyroscope. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

<b>gyro→get_lowestValue()</b>	<b>YGyro</b>
<b>gyro→lowestValue()gyro→get_lowestValue()</b>	

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

double **get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**gyro→get\_module()****YGyro****gyro→module()gyro→get\_module( )**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

<b>gyro→get_pitch()</b>	<b>YGyro</b>
<b>gyro→pitch()gyro→get_pitch( )</b>	

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**double get\_pitch( )**

L'axe de tangage peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe YRefFrame.

**Retourne :**

un nombre à virgule correspondant à l'assiette, exprimée en degrés (entre -90 et +90).

**gyro→get\_quaternionW()****YGyro****gyro→quaternionW()gyro→get\_quaternionW( )**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
double get_quaternionW( )
```

**Retourne :**

un nombre à virgule correspondant à la composante w du quaternion.

**gyro→get\_quaternionX()****YGyro****gyro→quaternionX()gyro→get\_quaternionX( )**

Retourne la composante `x` du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
double get_quaternionX( )
```

La composante `x` est essentiellement corrélée aux rotations sur l'axe de roulis.

**Retourne :**

un nombre à virgule correspondant à la composante `x` du quaternion.

---

**gyro→get\_quaternionY()****YGyro****gyro→quaternionY()gyro→get\_quaternionY( )**

Retourne la composante  $y$  du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
double get_quaternionY( )
```

La composante  $y$  est essentiellement corrélée aux rotations sur l'axe de tangage.

**Retourne :**

un nombre à virgule correspondant à la composante  $y$  du quaternion.

<b>gyro→get_quaternionZ()</b>	<b>YGyro</b>
<b>gyro→quaternionZ()gyro→get_quaternionZ( )</b>	

Retourne la composante  $z$  du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
double get_quaternionZ( )
```

La composante  $z$  est essentiellement corrélée aux rotations sur l'axe de lacet.

**Retourne :**

un nombre à virgule correspondant à la composante  $z$  du quaternion.

**gyro→get\_recordedData()****YGyro****gyro→recordedData()gyro→get\_recordedData( )**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

<b>gyro→get_reportFrequency()</b>	<b>YGyro</b>
<b>gyro→reportFrequency()</b>	<b>gyro→get_reportFrequency()</b>

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
string get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**gyro→get\_resolution()****YGyro****gyro→resolution()gyro→get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**gyro→get\_roll()**

**YGyro**

**gyro→roll()gyro→get\_roll( )**

---

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**double get\_roll( )**

L'axe de roulis peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

**Retourne :**

un nombre à virgule correspondant à l'inclinaison, exprimée en degrés (entre -180 et +180).

**gyro→get\_unit()****YGyro****gyro→unit()gyro→get\_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la vitesse angulaire est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**gyro→get(userData())**

**YGyro**

**gyro→userData()gyro→get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**gyro→get\_xValue()****YGyro****gyro→xValue()gyro→get\_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

```
double get_xValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_XVALUE\_INVALID.

<b>gyro→get_yValue()</b>	<b>YGyro</b>
<b>gyro→yValue()gyro→get_yValue( )</b>	

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

```
double get_yValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_YVALUE\_INVALID.

**gyro→get\_zValue()****YGyro****gyro→zValue()gyro→get\_zValue( )**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

```
double get_zValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_ZVALUE\_INVALID.

**gyro→isOnline()****YGyro**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le gyroscope est joignable, false sinon

**gyro→load()****YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→loadCalibrationPoints()** gyro→  
**loadCalibrationPoints( )**

**YGyro**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→nextGyro()****YGyro**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

`YGyro * nextGyro()`

**Retourne :**

un pointeur sur un objet `YGyro` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

<b>gyro→registerAnglesCallback()</b>	<b>YGyro</b>
<b>registerAnglesCallback( )</b>	

---

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
int registerAnglesCallback( YAnglesCallback callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelés trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter quatre arguments: l'objet YGyro du module qui a tourné, et les valeurs des trois angles roll, pitch et heading en degrés (nombres à virgules).

**gyro→registerQuaternionCallback()**  
**registerQuaternionCallback( )****YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
int registerQuaternionCallback( YQuatCallback callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appellé trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter cinq arguments: l'objet YGyro du module qui a tourné, et les valeurs des quatre composantes w, x, y et z du quaternion (nombres à virgules).

**gyro→registerTimedReportCallback()** gyro→  
**registerTimedReportCallback( )**

**YGyro**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YGyroTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**gyro→registerValueCallback()**  
**gyro→registerValueCallback( )**

**YGyro**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YGyroValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**gyro→set\_highestValue()**  
**gyro→setHighestValue()** gyro→  
**set\_highestValue( )**

---

**YGyro**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_logFrequency()**  
**gyro→setLogFrequency()** gyro→  
**set\_logFrequency( )**

YGyro

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>gyro→set_logicalName()</b>	<b>YGyro</b>
<b>gyro→setLogicalName()gyro→set_logicalName( )</b>	

---

Modifie le nom logique du gyroscope.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du gyroscope.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_lowestValue()****YGyro****gyro→setLowestValue()gyro→set\_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_reportFrequency()**  
**gyro→setReportFrequency()** gyro→  
**set\_reportFrequency( )**

**YGyro**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_resolution()****YGyro****gyro→setResolution()gyro→set\_resolution( )**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set(userData())** **YGyro**  
**gyro→setUserData()** **gyro→set(userData() )**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.19. Interface d'un port de Yocto-hub

Les objets YHubPort permettent de contrôler l'alimentation des ports d'un YoctoHub, ainsi que de détecter si un module y est raccordé et lequel. Un YHubPort reçoit toujours automatiquement comme nom logique le numéro de série unique du module Yoctopuce qui y est connecté.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_hubport.js'></script>
nodejs var yoctolib = require('yoctolib');
var YHubPort = yoctolib.YHubPort;
php require_once('yocto_hubport.php');
cpp #include "yocto_hubport.h"
m #import "yocto_hubport.h"
pas uses yocto_hubport;
vb yocto_hubport.vb
cs yocto_hubport.cs
java import com.yoctopuce.YoctoAPI.YHubPort;
py from yocto_hubport import *

```

### Fonction globales

#### yFindHubPort(func)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

#### yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

### Méthodes des objets YHubPort

#### hubport→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format TYPE ( NAME )=SERIAL.FUNCTIONID.

#### hubport→get\_advertisedValue()

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

#### hubport→get\_baudRate()

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

#### hubport→get\_enabled()

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

#### hubport→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

#### hubport→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

#### hubport→get\_friendlyName()

Retourne un identifiant global du port de Yocto-hub au format NOM\_MODULE.NOM\_FONCTION.

#### hubport→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### hubport→get\_functionId()

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

#### hubport→get\_hardwareId()

Retourne l'identifiant matériel unique du port de Yocto-hub au format SERIAL.FUNCTIONID.

#### hubport→get\_logicalName()

Retourne le nom logique du port de Yocto-hub.

**hubport→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**hubport→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**hubport→get\_portState()**

Retourne l'état actuel du port de Yocto-hub.

**hubport→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**hubport→isOnline()**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

**hubport→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

**hubport→load(msValidity)**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

**hubport→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

**hubport→nextHubPort()**

Continue l'énumération des port de Yocto-hub commencée à l'aide de yFirstHubPort( ).

**hubport→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**hubport→set\_enabled(newval)**

Modifie le mode d'activation du port du Yocto-hub.

**hubport→set\_logicalName(newval)**

Modifie le nom logique du port de Yocto-hub.

**hubport→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**hubport→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YHubPort.FindHubPort()****YHubPort****yFindHubPort()yFindHubPort( )**

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

**YHubPort\* yFindHubPort( const string& 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()**

**YHubPort**

### **yFirstHubPort()yFirstHubPort( )**

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

**YHubPort\* yFirstHubPort( )**

Utiliser la fonction `YHubPort.nextHubPort()` pour itérer sur les autres port de Yocto-hub.

**Retourne :**

un pointeur sur un objet `YHubPort`, correspondant au premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

**hubport->describe()****YHubPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le port de Yocto-hub (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**hubport→get\_advertisedValue()**  
**hubport→advertisedValue()** hubport→  
**get\_advertisedValue( )**

---

**YHubPort**

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

string **get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne **Y\_ADVERTISEDVALUE\_INVALID**.

**hubport→get\_baudRate()****YHubPort****hubport→baudRate()hubport→get\_baudRate( )**

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

**int get\_baudRate( )**

La valeur par défaut est 1000 kbps, une valeur inférieure révèle des problèmes de communication.

**Retourne :**

un entier représentant la vitesse de transfert utilisée par le port de Yocto-hub, en kbps

En cas d'erreur, déclenche une exception ou retourne **Y\_BAUDRATE\_INVALID**.

**hubport→get\_enabled()**

**YHubPort**

**hubport→enabled()hubport→get\_enabled( )**

---

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

**Y\_ENABLED\_enum 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()**  
**hubport→errorMessage()** hubport→  
**get\_errorMessage( )**

**YHubPort**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

**hubport→get\_errorType()**

**YHubPort**

**hubport→errorType()hubport→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

---

<b>hubport→get_friendlyName()</b>	<b>YHubPort</b>
<b>hubport→friendlyName()</b>	<b>hubport→</b>
<b>get_friendlyName( )</b>	

---

Retourne un identifiant global du port de Yocto-hub au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du port de Yocto-hub si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port de Yocto-hub (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**hubport→get\_functionDescriptor()**  
**hubport→functionDescriptor()** hubport→  
**get\_functionDescriptor( )**

---

**YHubPort**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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

---

<b>hubport→get_functionId()</b>	<b>YHubPort</b>
<b>hubport→functionId()hubport→get_functionId()</b>	

---

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

<b>hubport→get_hardwareId()</b>	<b>YHubPort</b>
<b>hubport→hardwareId()hubport→get_hardwareId( )</b>	

---

Retourne l'identifiant matériel unique du port de Yocto-hub au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port de Yocto-hub (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**hubport→get\_logicalName()**  
**hubport→logicalName()**  
**hubport→get\_logicalName( )**

**YHubPort**

Retourne le nom logique du port de Yocto-hub.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port de Yocto-hub. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

<b>hubport→get_module()</b>	<b>YHubPort</b>
<b>hubport→module()hubport→get_module( )</b>	

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* 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

---

<b>hubport→get_portState()</b>	<b>YHubPort</b>
<b>hubport→portState()hubport→get_portState()</b>	

---

Retourne l'état actuel du port de Yocto-hub.

**Y\_PORTSTATE\_enum get\_portState( )**

**Retourne :**

une valeur parmi Y\_PORTSTATE\_OFF, Y\_PORTSTATE\_OVRLD, Y\_PORTSTATE\_ON, Y\_PORTSTATE\_RUN et Y\_PORTSTATE\_PROG représentant l'état actuel du port de Yocto-hub

En cas d'erreur, déclenche une exception ou retourne Y\_PORTSTATE\_INVALID.

**hubport→get(userData)**

**YHubPort**

**hubport→userData()hubport→get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**hubport→isOnline()****YHubPort**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du port de Yocto-hub sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le port de Yocto-hub est joignable, false sinon

**hubport→load()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport→nextHubPort()**

**YHubPort**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

`YHubPort * nextHubPort()`

**Retourne :**

un pointeur sur un objet `YHubPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

<b>hubport→registerValueCallback()</b> <b>hubport→registerValueCallback( )</b>	<b>YHubPort</b>
-----------------------------------------------------------------------------------	-----------------

---

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YHubPortValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**hubport→set\_enabled()****YHubPort****hubport→setEnabled()hubport→set\_enabled( )**

Modifie le mode d'activation du port du Yocto-hub.

```
int set_enabled( Y_ENABLED_enum 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.

---

<b>hubport→set_logicalName()</b>	<b>YHubPort</b>
<b>hubport→setLogicalName()</b> hubport→ <b>set_logicalName( )</b>	

---

Modifie le nom logique du port de Yocto-hub.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du port de Yocto-hub.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport→set(userData)****YHubPort****hubport→setUserData()hubport→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.20. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_humidity.js'></script>
nodejs var yoctolib = require('yoctolib');
var YHumidity = yoctolib.YHumidity;
require_once('yocto_humidity.php');
#include "yocto_humidity.h"
m #import "yocto_humidity.h"
pas uses yocto_humidity;
vb yocto_humidity.vb
cs yocto_humidity.cs
java import com.yoctopuce.YoctoAPI.YHumidity;
py from yocto_humidity import *

```

### Fonction globales

#### yFindHumidity(func)

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

#### yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

### Méthodes des objets YHumidity

#### humidity→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### humidity→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### humidity→get\_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

#### humidity→get\_currentRawValue()

Retourne la valeur brute rentrée par le capteur (sans arrondi ni calibration).

#### humidity→get\_currentValue()

Retourne la mesure actuelle de l'humidité.

#### humidity→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### humidity→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### humidity→get\_friendlyName()

Retourne un identifiant global du capteur d'humidité au format NOM\_MODULE . NOM\_FONCTION.

#### humidity→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### humidity→get\_functionId()

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

#### humidity→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.
<b>humidity→get_highestValue()</b>
Retourne la valeur maximale observée pour l'humidité.
<b>humidity→get_logFrequency()</b>
Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>humidity→get_logicalName()</b>
Retourne le nom logique du capteur d'humidité.
<b>humidity→get_lowestValue()</b>
Retourne la valeur minimale observée pour l'humidité.
<b>humidity→get_module()</b>
Retourne l'objet 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_recordedData(startTime, endTime)</b>
Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>humidity→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>humidity→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>humidity→get_unit()</b>
Retourne l'unité dans laquelle l'humidité est exprimée.
<b>humidity→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>humidity→isOnline()</b>
Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.
<b>humidity→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.
<b>humidity→load(msValidity)</b>
Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.
<b>humidity→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>humidity→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.
<b>humidity→nextHumidity()</b>
Continue l'énumération des capteurs d'humidité commencée à l'aide de yFirstHumidity( ).
<b>humidity→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>humidity→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>humidity→set_highestValue(newval)</b>
Modifie la mémoire de valeur maximale observée pour l'humidité.
<b>humidity→set_logFrequency(newval)</b>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**humidity→set\_logicalName(newval)**

Modifie le nom logique du capteur d'humidité.

**humidity→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour l'humidité.

**humidity→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**humidity→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**humidity→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**humidity→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YHumidity.FindHumidity()****YHumidity****yFindHumidity()yFindHumidity( )**

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

**YHumidity\* yFindHumidity( const string& 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()**

**YHumidity**

### **yFirstHumidity()yFirstHumidity( )**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

**YHumidity\* yFirstHumidity( )**

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

**Retourne :**

un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

**humidity→calibrateFromPoints()humidity→  
calibrateFromPoints( )****YHumidity**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**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()****YHumidity**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE (NAME )=SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur d'humidité (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**humidity→get\_advertisedValue()**  
**humidity→advertisedValue()**humidity→  
**get\_advertisedValue( )**

**YHumidity**

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur d'humidité (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**humidity→get\_currentRawValue()** YHumidity  
**humidity→currentRawValue()** **humidity→get\_currentRawValue( )**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
double get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**humidity→get\_currentValue()**  
**humidity→currentValue()humidity→**  
**get\_currentValue()**

**YHumidity**

Retourne la mesure actuelle de l'humidité.

**double get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la mesure actuelle de l'humidité

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**humidity→getErrorMessage()**

**YHumidity**

**humidity→errorMessage()** **humidity→**  
**getErrorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

**humidity→get\_errorType()****YHumidity****humidity→errorType()humidity→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

---

<b>humidity→get_friendlyName()</b>	<b>YHumidity</b>
<b>humidity→friendlyName()</b> humidity→ <b>get_friendlyName( )</b>	

Retourne un identifiant global du capteur d'humidité au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur d'humidité si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur d'humidité (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**humidity→get\_functionDescriptor()**  
**humidity→functionDescriptor()** **humidity→get\_functionDescriptor( )**

**YHumidity**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>humidity→get_functionId()</b>	<b>YHumidity</b>
<b>humidity→functionId()</b> <b>humidity→get_functionId( )</b>	

---

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**humidity→get\_hardwareId()****YHumidity****humidity→hardwareId()humidity→  
get\_hardwareId( )**

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur d'humidité (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**humidity→get\_highestValue()**  
**humidity→highestValue()**humidity→  
**get\_highestValue( )**

---

YHumidity

Retourne la valeur maximale observée pour l'humidité.

double **get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'humidité

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**humidity→get\_logFrequency()**  
**humidity→logFrequency()**humidity→  
**get\_logFrequency( )**

**YHumidity**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**humidity→get\_logicalName()**

**YHumidity**

**humidity→logicalName()humidity→**

**get\_logicalName( )**

---

Retourne le nom logique du capteur d'humidité.

string **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()**  
**humidity→lowestValue()**humidity→  
**get\_lowestValue( )**

**YHumidity**

Retourne la valeur minimale observée pour l'humidité.

**double get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'humidité

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**humidity→get\_module()**

**YHumidity**

**humidity→module()humidity→get\_module()**

---

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**humidity→get\_recordedData()**  
**humidity→recordedData()humidity→**  
**get\_recordedData()**

**YHumidity**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**humidity→get\_reportFrequency()** YHumidity  
**humidity→reportFrequency()** **humidity→get\_reportFrequency( )**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**humidity→get\_resolution()**  
**humidity→resolution()** **humidity→get\_resolution( )**

**YHumidity**

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne **Y\_RESOLUTION\_INVALID**.

**humidity→get\_unit()**

**YHumidity**

**humidity→unit()humidity→get\_unit()**

---

Retourne l'unité dans laquelle l'humidité est exprimée.

string **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'humidité est exprimée

En cas d'erreur, déclenche une exception ou retourne **Y\_UNIT\_INVALID**.

**humidity→get(userData)****YHumidity****humidity→userData()humidity→get(userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * get(userData)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**humidity→isOnline()****YHumidity**

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du capteur d'humidité sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur d'humidité est joignable, false sinon

**humidity→load()****YHumidity**

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→loadCalibrationPoints()humidity→  
loadCalibrationPoints()****YHumidity**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→nextHumidity()**  
**humidity→nextHumidity( )**

**YHumidity**

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity( )`.

`YHumidity * nextHumidity( )`

**Retourne :**

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YHumidityTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

## Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

**humidity→registerValueCallback()**  
**humidity→registerValueCallback( )****YHumidity**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YHumidityValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**humidity→set\_highestValue()** YHumidity  
**humidity→setHighestValue()** **humidity→set\_highestValue( )**

---

Modifie la mémoire de valeur maximale observée pour l'humidité.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_logFrequency()****YHumidity****humidity→setLogFrequency()humidity→  
set\_logFrequency( )**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( const string& newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>humidity→set_logicalName()</b>	<b>YHumidity</b>
<b>humidity→setLogicalName()</b> <b>humidity→set_logicalName( )</b>	

Modifie le nom logique du capteur d'humidité.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur d'humidité.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_lowestValue()**

**YHumidity**

**humidity→setLowestValue()** **humidity→set\_lowestValue( )**

Modifie la mémoire de valeur minimale observée pour l'humidité.

**int set\_lowestValue( double newval)**

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_reportFrequency()** YHumidity  
**humidity→setReportFrequency()** **humidity→set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_resolution()**  
**humidity→setResolution()** **humidity→**  
**set\_resolution( )**

**YHumidity**

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set(userData)**

**YHumidity**

**humidity→setUserData()humidity→  
set(userData)()**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.21. Interface de la fonction Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la led à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_led.js'></script>
nodejs var yoctolib = require('yoctolib');
var YLed = yoctolib.YLed;
php require_once('yocto_led.php');
cpp #include "yocto_led.h"
m #import "yocto_led.h"
pas uses yocto_led;
vb yocto_led.vb
cs yocto_led.cs
java import com.yoctopuce.YoctoAPI.YLed;
py from yocto_led import *

```

### Fonction globales

#### **yFindLed(func)**

Permet de retrouver une led d'après un identifiant donné.

#### **yFirstLed()**

Commence l'énumération des leds accessibles par la librairie.

### Méthodes des objets YLed

#### **led->describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **led->get\_advertisedValue()**

Retourne la valeur courante de la led (pas plus de 6 caractères).

#### **led->get\_blinking()**

Retourne le mode de signalisation de la led.

#### **led->get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

#### **led->get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

#### **led->get\_friendlyName()**

Retourne un identifiant global de la led au format NOM\_MODULE . NOM\_FONCTION.

#### **led->get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **led->get\_functionId()**

Retourne l'identifiant matériel de la led, sans référence au module.

#### **led->get\_hardwareId()**

Retourne l'identifiant matériel unique de la led au format SERIAL . FUNCTIONID.

#### **led->get\_logicalName()**

Retourne le nom logique de la led.

#### **led->get\_luminosity()**

Retourne l'intensité de la led en pour cent.

#### **led->get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>led-&gt;get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>led-&gt;get_power()</b>
Retourne l'état courant de la led.
<b>led-&gt;get_userData()</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>led-&gt;isOnline()</b>
Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.
<b>led-&gt;isOnline_async(callback, context)</b>
Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.
<b>led-&gt;load(msValidity)</b>
Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.
<b>led-&gt;load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.
<b>led-&gt;nextLed()</b>
Continue l'énumération des leds commencée à l'aide de yFirstLed( ).
<b>led-&gt;registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>led-&gt;set_blinking(newval)</b>
Modifie le mode de signalisation de la led.
<b>led-&gt;set_logicalName(newval)</b>
Modifie le nom logique de la led.
<b>led-&gt;set_luminosity(newval)</b>
Modifie l'intensité lumineuse de la led (en pour cent).
<b>led-&gt;set_power(newval)</b>
Modifie l'état courant de la led.
<b>led-&gt;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>led-&gt;wait_async(callback, context)</b>
Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YLed.FindLed()****YLed****yFindLed()yFindLed( )**

Permet de retrouver une led d'après un identifiant donné.

**YLed\* yFindLed( const string& 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()

YLed

### yFirstLed()yFirstLed( )

Commence l'énumération des leds accessibles par la librairie.

YLed\* **yFirstLed( )**

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()****YLed**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant la led (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**led→get\_advertisedValue()**  
**led→advertisedValue()** led→  
**get\_advertisedValue( )**

---

YLed

Retourne la valeur courante de la led (pas plus de 6 caractères).

string **get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**led→get\_blinking()****YLed****led→blinking()led→get\_blinking()**

Retourne le mode de signalisation de la led.

**Y\_BLINKING\_enum get\_blinking( )****Retourne :**

une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

En cas d'erreur, déclenche une exception ou retourne Y\_BLINKING\_INVALID.

**led→get\_errorMessage()**

YLed

**led→errorMessage()led→get\_errorMessage( )**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la led.

**led->get\_errorType()****YLed****led->errorType()led->get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la led.

**led→get\_friendlyName()** YLed  
**led→friendlyName()** led→get\_friendlyName( )

---

Retourne un identifiant global de la led au format NOM\_MODULE . NOM\_FONCTION.

```
string get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la led si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la led (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant la led en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**led→get\_functionDescriptor()  
led→functionDescriptor()led→  
get\_functionDescriptor( )**

**YLed**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**led→get\_functionId()**

YLed

**led→functionId()led→get\_functionId( )**

---

Retourne l'identifiant matériel de la led, sans référence au module.

string **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la led (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**led→get\_hardwareId()****YLed****led→hardwareId()led→get\_hardwareId( )**

Retourne l'identifiant matériel unique de la led au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la led (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**led→get\_logicalName()**

YLed

**led→logicalName()led→get\_logicalName( )**

---

Retourne le nom logique de la led.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique de la led. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**led→get\_luminosity()****YLed****led→luminosity()led→get\_luminosity()**

Retourne l'intensité de la led en pour cent.

```
int get_luminosity( )
```

**Retourne :**

un entier représentant l'intensité de la led en pour cent

En cas d'erreur, déclenche une exception ou retourne Y\_LUMINOSITY\_INVALID.

**led→get\_module()**

**YLed**

**led→module()led→get\_module()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

---

**led→get\_power()**  
**led→power()led→get\_power( )**

---

YLed

Retourne l'état courant de la led.

**Y\_POWER\_enum get\_power( )**

**Retourne :**

soit Y\_POWER\_OFF, soit Y\_POWER\_ON, selon l'état courant de la led

En cas d'erreur, déclenche une exception ou retourne Y\_POWER\_INVALID.

**led→get(userData)**

YLed

**led→userData()led→get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**led->isOnline()|led->isOnline( )****YLed**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de la led sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si la led est joignable, false sinon

**led→load()**

YLed

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led->nextLed()****YLed**

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

`YLed * nextLed()`

**Retourne :**

un pointeur sur un objet `YLed` accessible en ligne, ou `null` lorsque l'énumération est terminée.

```
led->registerValueCallback() led->  
registerValueCallback()
```

YLed

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YLedValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**led->set\_blinking()****YLed****led->setBlinking()led->set\_blinking( )**

Modifie le mode de signalisation de la led.

```
int set_blinking( Y_BLINKING_enum newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led->set\_logicalName()** YLed  
**led->setLogicalName()** led->set\_logicalName( )

---

Modifie le nom logique de la led.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la led.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→set\_luminosity()****YLed****led→setLuminosity()led→set\_luminosity( )**

Modifie l'intensité lumineuse de la led (en pour cent).

```
int set_luminosity( int 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()**  
**led->setPower()**  
**led->set\_power( )**

---

YLed

Modifie l'état courant de la led.

```
int set_power( Y_POWER_enum newval)
```

**Paramètres :**

**newval** soit Y\_POWER\_OFF, soit Y\_POWER\_ON, selon l'état courant de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→set(userData)****YLed****led→setUserData()|led→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.22. Interface de la fonction LightSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_lightsensor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YLightSensor = yoctolib.YLightSensor;
php require_once('yocto_lightsensor.php');
cpp #include "yocto_lightsensor.h"
m #import "yocto_lightsensor.h"
pas uses yocto_lightsensor;
vb yocto_lightsensor.vb
cs yocto_lightsensor.cs
java import com.yoctopuce.YoctoAPI.YLightSensor;
py from yocto_lightsensor import *

```

### Fonction globales

#### yFindLightSensor(func)

Permet de retrouver un capteur de lumière d'après un identifiant donné.

#### yFirstLightSensor()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

### Méthodes des objets YLightSensor

#### lightsensor→calibrate(calibratedVal)

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

#### lightsensor→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### lightsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### lightsensor→get\_advertisedValue()

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

#### lightsensor→get\_currentRawValue()

Retourne la valeur brute rentrée par le capteur (sans arrondi ni calibration).

#### lightsensor→get\_currentValue()

Retourne la mesure actuelle de la lumière ambiante.

#### lightsensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

#### lightsensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

#### lightsensor→get\_friendlyName()

Retourne un identifiant global du capteur de lumière au format NOM\_MODULE . NOM\_FONCTION.

#### lightsensor→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### lightsensor→get\_functionId()

	Retourne l'identifiant matériel du capteur de lumière, sans référence au module.
<b>lightsensor→get_hardwareId()</b>	Retourne l'identifiant matériel unique du capteur de lumière au format SERIAL . FUNCTIONID.
<b>lightsensor→get_highestValue()</b>	Retourne la valeur maximale observée pour la lumière ambiante.
<b>lightsensor→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>lightsensor→get_logicalName()</b>	Retourne le nom logique du capteur de lumière.
<b>lightsensor→get_lowestValue()</b>	Retourne la valeur minimale observée pour la lumière ambiante.
<b>lightsensor→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>lightsensor→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>lightsensor→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>lightsensor→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>lightsensor→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>lightsensor→get_unit()</b>	Retourne l'unité dans laquelle la lumière ambiante est exprimée.
<b>lightsensor→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>lightsensor→isOnline()</b>	Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.
<b>lightsensor→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.
<b>lightsensor→load(msValidity)</b>	Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.
<b>lightsensor→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>lightsensor→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.
<b>lightsensor→nextLightSensor()</b>	Continue l'énumération des capteurs de lumière commencée à l'aide de yFirstLightSensor( ).
<b>lightsensor→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>lightsensor→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>lightsensor→set_highestValue(newval)</b>	

### 3. Reference

---

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

**[lightsensor→set\\_logFrequency\(newval\)](#)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**[lightsensor→set\\_logicalName\(newval\)](#)**

Modifie le nom logique du capteur de lumière.

**[lightsensor→set\\_lowestValue\(newval\)](#)**

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

**[lightsensor→set\\_reportFrequency\(newval\)](#)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**[lightsensor→set\\_resolution\(newval\)](#)**

Modifie la résolution des valeurs physique mesurées.

**[lightsensor→set\(userData\)](#)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**[lightsensor→wait\\_async\(callback, context\)](#)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YLightSensor.FindLightSensor()****YLightSensor****yFindLightSensor()yFindLightSensor( )**

Permet de retrouver un capteur de lumière d'après un identifiant donné.

**YLightSensor\* yFindLightSensor( const string& 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() yFirstLightSensor()yFirstLightSensor( )**

---

**YLightSensor**

Commence l'énumération des capteurs de lumière accessibles par la librairie.

**YLightSensor\* yFirstLightSensor( )**

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

**Retourne :**

un pointeur sur un objet `YLightSensor`, correspondant au premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

**lightsensor→calibrate()****YLightSensor**

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

```
int calibrate( double calibratedVal)
```

**Paramètres :**

**calibratedVal** la consigne de valeur désirée.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→calibrateFromPoints()**  
**lightsensor→calibrateFromPoints( )****YLightSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**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()****YLightSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de lumière (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**lightsensor→get\_advertisedValue()** YLightSensor  
**lightsensor→advertisedValue()** lightsensor→  
**get\_advertisedValue( )**

---

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

string **get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de lumière (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

**lightsensor→get\_currentRawValue()** YLightSensor  
**lightsensor→currentRawValue()** lightsensor→  
**get\_currentRawValue( )**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
double get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**lightsensor→get\_currentValue()**  
**lightsensor→currentValue()****lightsensor→get\_currentValue( )**

---

**YLightSensor**

Retourne la mesure actuelle de la lumière ambiante.

double **get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la mesure actuelle de la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTVALUE\_INVALID**.

---

**lightsensor→get\_errorMessage()**  
**lightsensor→errorMessage()****lightsensor→get\_errorMessage( )**

**YLightSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
string get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

**lightsensor→get\_errorType()**  
**lightsensor→errorType()lightsensor→**  
**get\_errorType( )**

**YLightSensor**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

**lightsensor→get\_friendlyName()****YLightSensor****lightsensor→friendlyName() lightsensor→  
get\_friendlyName( )**

Retourne un identifiant global du capteur de lumière au format NOM\_MODULE . NOM\_FONCTION.

```
string get_friendlyName( )
```

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de lumière si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de lumière (par exemple: MyCustomName . relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière en utilisant les noms logiques (ex: MyCustomName . relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

<b>lightsensor→get_functionDescriptor()</b>	<b>YLightSensor</b>
<b>lightsensor→functionDescriptor()lightsensor→get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**lightsensor→get\_functionId()****YLightSensor****lightsensor→functionId()** lightsensor→  
**get\_functionId()**

---

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

```
string get_functionId()
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

**lightsensor→get\_hardwareId()**

**YLightSensor**

**lightsensor→hardwareId()** lightsensor→  
**get\_hardwareId( )**

---

Retourne l'identifiant matériel unique du capteur de lumière au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de lumière (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**lightsensor→get\_highestValue()****YLightSensor****lightsensor→highestValue()lightsensor→  
get\_highestValue()**

Retourne la valeur maximale observée pour la lumière ambiante.

```
double get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**lightsensor→get\_logFrequency()** YLightSensor  
**lightsensor→logFrequency()** lightsensor→  
**get\_logFrequency( )**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

string **get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**lightsensor→get\_logicalName()**  
**lightsensor→logicalName()** lightsensor→  
**get\_logicalName( )**

**YLightSensor**

Retourne le nom logique du capteur de lumière.

```
string 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()**  
**lightsensor→lowestValue()****lightsensor→get\_lowestValue()**

**YLightSensor**

Retourne la valeur minimale observée pour la lumière ambiante.

```
double get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne **Y\_LOWESTVALUE\_INVALID**.

---

```
lightsensor->get_module()  
lightsensor->module()lightsensor->  
get_module()
```

**YLightSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
YModule * get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**lightsensor→get\_recordedData()**  
**lightsensor→recordedData() lightsensor→**  
**get\_recordedData( )**

**YLightSensor**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

<b>lightsensor→get_reportFrequency()</b>	<b>YLightSensor</b>
<b>lightsensor→reportFrequency()</b> lightsensor→ <b>get_reportFrequency( )</b>	

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
string get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**lightsensor→get\_resolution()**  
**lightsensor→resolution()lightsensor→**  
**get\_resolution()**

---

**YLightSensor**

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**lightsensor→get\_unit()****YLightSensor****lightsensor→unit() lightsensor→get\_unit( )**

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la lumière ambiante est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**lightsensor→get(userData)**  
**lightsensor→userData()** lightsensor→  
**get(userData)**

**YLightSensor**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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()****YLightSensor**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du capteur de lumière sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de lumière est joignable, false sinon

**lightsensor→load()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→loadCalibrationPoints()**  
**lightsensor→  
loadCalibrationPoints( )****YLightSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→nextLightSensor()**  
**nextLightSensor( )**

**YLightSensor**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

**YLightSensor \* nextLightSensor( )**

**Retourne :**

un pointeur sur un objet `YLightSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**lightsensor→registerTimedReportCallback()****YLightSensor****lightsensor→registerTimedReportCallback( )**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YLightSensorTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**lightsensor→registerValueCallback()** **lightsensor→registerValueCallback( )**

**YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**int registerValueCallback( YLightSensorValueCallback **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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**lightsensor→set\_highestValue()****YLightSensor****lightsensor→setHighestValue()**  
**lightsensor→set\_highestValue()**

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la lumière ambiante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logFrequency()** **YLightSensor**  
**lightsensor→setLogFrequency()** **lightsensor→**  
**set\_logFrequency( )**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logicalName()**  
**lightsensor→setLogicalName()** lightsensor→  
**set\_logicalName( )**

**YLightSensor**

Modifie le nom logique du capteur de lumière.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de lumière.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_lowestValue()** YLightSensor  
**lightsensor→setLowestValue()** lightsensor→  
**set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la lumière ambiante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_reportFrequency()****YLightSensor****lightsensor→setReportFrequency() lightsensor→  
set\_reportFrequency( )**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( const string& newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_resolution()** YLightSensor  
**lightsensor→setResolution()** **lightsensor→set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set(userData)**  
**lightsensor→setUserData()****lightsensor→**  
**set(userData)**

**YLightSensor**

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).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.23. Interface de la fonction Magnetometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_magnetometer.js'></script>
nodejs var yoctolib = require('yoctolib');
var YMagnetometer = yoctolib.YMagnetometer;
php require_once('yocto_magnetometer.php');
cpp #include "yocto_magnetometer.h"
m #import "yocto_magnetometer.h"
pas uses yocto_magnetometer;
vb yocto_magnetometer.vb
cs yocto_magnetometer.cs
java import com.yoctopuce.YoctoAPI.YMagnetometer;
py from yocto_magnetometer import *

```

### Fonction globales

#### **yFindMagnetometer(func)**

Permet de retrouver un magnétomètre d'après un identifiant donné.

#### **yFirstMagnetometer()**

Commence l'énumération des magnétomètres accessibles par la librairie.

### Méthodes des objets YMagnetometer

#### **magnetometer→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **magnetometer→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **magnetometer→get\_advertisedValue()**

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

#### **magnetometer→get\_currentRawValue()**

Retourne la valeur brute renournée par le capteur (sans arrondi ni calibration).

#### **magnetometer→get\_currentValue()**

Retourne la valeur actuelle du champ magnétique.

#### **magnetometer→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### **magnetometer→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### **magnetometer→get\_friendlyName()**

Retourne un identifiant global du magnétomètre au format NOM\_MODULE . NOM\_FONCTION.

#### **magnetometer→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **magnetometer→get\_functionId()**

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

#### **magnetometer→get\_hardwareId()**

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL . FUNCTIONID.

<b>magnetometer→get_highestValue()</b>	Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.
<b>magnetometer→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>magnetometer→get_logicalName()</b>	Retourne le nom logique du magnétomètre.
<b>magnetometer→get_lowestValue()</b>	Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.
<b>magnetometer→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>magnetometer→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>magnetometer→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>magnetometer→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>magnetometer→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>magnetometer→get_unit()</b>	Retourne l'unité dans laquelle le champ magnétique est exprimée.
<b>magnetometer→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>magnetometer→get_xValue()</b>	Retourne la composante X du champ magnétique, sous forme de nombre à virgule.
<b>magnetometer→get_yValue()</b>	Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.
<b>magnetometer→get_zValue()</b>	Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.
<b>magnetometer→isOnline()</b>	Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.
<b>magnetometer→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.
<b>magnetometer→load(msValidity)</b>	Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.
<b>magnetometer→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>magnetometer→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.
<b>magnetometer→nextMagnetometer()</b>	Continue l'énumération des magnétomètres commencée à l'aide de yFirstMagnetometer( ).
<b>magnetometer→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.

### 3. Reference

#### **magnetometer→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **magnetometer→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

#### **magnetometer→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### **magnetometer→set\_logicalName(newval)**

Modifie le nom logique du magnétomètre.

#### **magnetometer→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

#### **magnetometer→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **magnetometer→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **magnetometer→set(userData,data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **magnetometer→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YMagnetometer.FindMagnetometer()****YMagnetometer****yFindMagnetometer()yFindMagnetometer( )**

Permet de retrouver un magnétomètre d'après un identifiant donné.

**YMagnetometer\* yFindMagnetometer( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le magnétomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMagnetometer.isOnLine()` pour tester si le magnétomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le magnétomètre sans ambiguïté

**Retourne :**

un objet de classe `YMagnetometer` qui permet ensuite de contrôler le magnétomètre.

## YMagnetometer.FirstMagnetometer()

## YMagnetometer

### yFirstMagnetometer()yFirstMagnetometer( )

Commence l'énumération des magnétomètres accessibles par la librairie.

YMagnetometer\* **yFirstMagnetometer( )**

Utiliser la fonction YMagnetometer.nextMagnetometer() pour itérer sur les autres magnétomètres.

**Retourne :**

un pointeur sur un objet YMagnetometer, correspondant au premier magnétomètre accessible en ligne, ou null si il n'y a pas de magnétomètres disponibles.

**magnetometer→calibrateFromPoints()****YMagetometer****magnetometer→calibrateFromPoints( )**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→describe()**  
**magnetometer→  
describe( )**

**YMagnetometer**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format TYPE ( NAME )=SERIAL . FUNCTIONID.

string **describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le magnétomètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

---

<b>magnetometer→get_advertisedValue()</b>	<b>YMagnetometer</b>
<b>magnetometer→advertisedValue()</b> <b>magnetometer→get_advertisedValue( )</b>	

---

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

```
string get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du magnétomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**magnetometer→get\_currentRawValue()**

**YMagnetometer**

**magnetometer→currentRawValue()magnetometer→  
get\_currentRawValue( )**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

**double get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

---

**magnetometer→get\_currentValue()** YMagnetometer  
**magnetometer→currentValue()magnetometer→**  
**get\_currentValue()**

---

Retourne la valeur actuelle du champ magnétique.

**double get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur actuelle du champ magnétique

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**magnetometer→get\_errorMessage()**

**YMagnetometer**

**magnetometer→errorMessage()magnetometer→  
get\_errorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

**magnetometer→get\_errorType()**  
**magnetometer→errorType()****magnetometer→get\_errorType( )**

**YMagnetometer**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

**magnetometer→get\_friendlyName()****YMagnetometer****magnetometer→friendlyName()magnetometer→  
get\_friendlyName( )**

Retourne un identifiant global du magnétomètre au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du magnétomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du magnétomètre (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le magnétomètre en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**magnetometer→get\_functionDescriptor()**

**YMagnetometer**

**magnetometer→functionDescriptor()magnetometer**

**→get\_functionDescriptor( )**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**magnetometer→get\_functionId()**

**YMagnetometer**

**magnetometer→functionId()magnetometer→  
get\_functionId( )**

---

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

<b>magnetometer→get_hardwareId()</b>	<b>YMagnetometer</b>
<b>magnetometer→hardwareId()magnetometer→get_hardwareId( )</b>	

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du magnétomètre (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**magnetometer→get\_highestValue()**

**YMagnetometer**

**magnetometer→highestValue()****magnetometer→**

**get\_highestValue( )**

---

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

**double get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

---

<b>magnetometer→get_logFrequency()</b>	<b>YMagnetometer</b>
<b>magnetometer→logFrequency()magnetometer→get_logFrequency( )</b>	

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
string get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**magnetometer→get\_logicalName()**

**YMagnetometer**

**magnetometer→logicalName()magnetometer→  
get\_logicalName( )**

---

Retourne le nom logique du magnétomètre.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du magnétomètre. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**magnetometer→get\_lowestValue()****YMagnetometer****magnetometer→lowestValue()magnetometer→  
get\_lowestValue( )**

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

**double get\_lowestValue( )****Retourne :**

une valeur numérique représentant la valeur minimale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**magnetometer→get\_module()**

**YMagnetometer**

**magnetometer→module()magnetometer→  
get\_module( )**

---

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**magnetometer→get\_recordedData()**

**YMagnetometer**

**magnetometer→recordedData()magnetometer→  
get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**magnetometer→get\_reportFrequency()**

**YMagnetometer**

**magnetometer→reportFrequency()magnetometer→**

**get\_reportFrequency( )**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**magnetometer→get\_resolution()**

**YMagnetometer**

**magnetometer→resolution()magnetometer→  
get\_resolution( )**

---

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne **Y\_RESOLUTION\_INVALID**.

**magnetometer→get\_unit()**

**YMagnetometer**

**magnetometer→unit()magnetometer→get\_unit( )**

---

Retourne l'unité dans laquelle le champ magnétique est exprimée.

**string get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le champ magnétique est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**magnetometer→get(userData)**

**YMagnetometer**

**magnetometer→userData()magnetometer→**  
**get(userData( )**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* get(userData( )**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**magnetometer→get\_xValue()**

**YMagnetometer**

**magnetometer→xValue()magnetometer→  
get\_xValue( )**

---

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

**double get\_xValue( )**

**Retourne :**

une valeur numérique représentant la composante X du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_XVALUE\_INVALID**.

**magnetometer→get\_yValue()** **YMagnetometer**  
**magnetometer→yValue()magnetometer→**  
**get\_yValue( )**

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

**double get\_yValue( )**

**Retourne :**

une valeur numérique représentant la composante Y du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_YVALUE\_INVALID**.

**magnetometer→get\_zValue()**

**YMagnetometer**

**magnetometer→zValue()magnetometer→  
get\_zValue( )**

---

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

**double get\_zValue( )**

**Retourne :**

une valeur numérique représentant la composante Z du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_ZVALUE\_INVALID**.

**magnetometer**→**isOnline()****magnetometer**→  
**isOnline( )**

**YMagnetometer**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du magnétomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le magnétomètre est joignable, false sinon

**magnetometer→load()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→loadCalibrationPoints()**  
**magnetometer→loadCalibrationPoints( )**

**YMagnetometer**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer** → **nextMagnetometer()** **magnetometer**  
→ **nextMagnetometer( )**

**YMagnetometer**

Continue l'énumération des magnétomètres commencée à l'aide de **yFirstMagnetometer( )**.

**YMagnetometer \* nextMagnetometer( )**

**Retourne :**

un pointeur sur un objet **YMagnetometer** accessible en ligne, ou **null** lorsque l'énumération est terminée.

**magnetometer→registerTimedReportCallback()**

**YMagnetometer**

**magnetometer→**

**registerTimedReportCallback( )**

---

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**int registerTimedReportCallback( YMagnetometerTimedReportCallback callback)**

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**magnetometer→registerValueCallback()****YMagnetometer****magnetometer→registerValueCallback( )**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YMagnetometerValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**magnetometer→set\_highestValue()**

**YMagnetometer**

**magnetometer→setHighestValue()magnetometer→  
set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_logFrequency()**  
**magnetometer→setLogFrequency()****magnetometer→set\_logFrequency( )**

**YMagnetometer**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>magnetometer→set_logicalName()</b>	<b>YMagnetometer</b>
<b>magnetometer→setLogicalName()</b> <b>magnetometer→</b> <b>set_logicalName( )</b>	

Modifie le nom logique du magnétomètre.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du magnétomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_lowestValue()** YMagnetometer  
**magnetometer→setLowestValue()** ~~magnetometer→~~  
**set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_reportFrequency()**  
**magnetometer→setReportFrequency()**  
**magnetometer→set\_reportFrequency( )**

**YMagnetometer**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_resolution()**  
**magnetometer→setResolution()****magnetometer→set\_resolution()**

**YMagnetometer**

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set(userData)**

**YMagnetometer**

**magnetometer→setUserData()****magnetometer→**  
**set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.24. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Méthodes des objets YMeasure

#### **measure→get\_averageValue()**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

#### **measure→get\_maxValue()**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_minValue()**

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_startTimeUTC()**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**measure→get\_averageValue()**  
**measure→averageValue()** measure→  
**get\_averageValue()**

**YMeasure**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

double **get\_averageValue( )**

**Retourne :**

un nombre décimal correspondant à la valeur moyenne observée.

<b>measure→get_endTimeUTC()</b>	<b>YMeasure</b>
<b>measure→endTimeUTC()</b> measure→ <b>get_endTimeUTC( )</b>	

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**double get\_endTimeUTC( )**

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclurent une fraction décimale.

**Retourne :**

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

---

**measure→get\_maxValue()****YMeasure****measure→maxValue()measure→get\_maxValue( )**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

```
double get_maxValue( )
```

**Retourne :**

un nombre décimal correspondant à la plus grande valeur observée.

**measure→get\_minValue()**

**YMeasure**

**measure→minValue()measure→get\_minValue()**

---

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

double **get\_minValue( )**

**Retourne :**

un nombre décimal correspondant à la plus petite valeur observée.

**measure→getStartTimeUTC()**  
**measure→startTimeUTC()measure→**  
**getStartTimeUTC()**

**YMeasure**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**double getStartTimeUTC( )**

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclurent une fraction décimale.

**Retourne :**

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la début de la mesure.

## 3.25. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Fonction globales

#### **yFindModule(func)**

Permet de retrouver un module d'après son numéro de série ou son nom logique.

#### **yFirstModule()**

Commence l'énumération des modules accessibles par la librairie.

### Méthodes des objets YModule

#### **module→describe()**

Retourne un court texte décrivant le module.

#### **module→download(pathname)**

Télécharge le fichier choisi du module et retourne son contenu.

#### **module→functionCount()**

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

#### **module→functionId(functionIndex)**

Retourne l'identifiant matériel de la *n*ième fonction du module.

#### **module→functionName(functionIndex)**

Retourne le nom logique de la *n*ième fonction du module.

#### **module→functionValue(functionIndex)**

Retourne la valeur publiée par la *n*ième fonction du module.

#### **module→get\_beacon()**

Retourne l'état de la balise de localisation.

#### **module→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

#### **module→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

#### **module→get\_firmwareRelease()**

Retourne la version du logiciel embarqué du module.

#### **module→get\_hardwareId()**

Retourne l'identifiant unique du module.

#### **module→get\_icon2d()**

Retourne l'icône du module.
<b>module→get_lastLogs()</b>
Retourne une chaîne de caractère contenant les derniers logs 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>
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
<b>module→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
<b>module→nextModule()</b>
Continue l'énumération des modules commencée à l'aide de yFirstModule( ).
<b>module→reboot(secBeforeReboot)</b>
Agende un simple redémarrage du module dans un nombre donné de secondes.
<b>module→registerLogCallback(callback)</b>
todo
<b>module→revertFromFlash()</b>
Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.
<b>module→saveToFlash()</b>

### 3. Reference

---

Sauve les réglages courants dans la mémoire non volatile du module.

**module→set\_beacon(newval)**

Allume ou éteint la balise de localisation du module.

**module→set\_logicalName(newval)**

Change le nom logique du module.

**module→set\_luminosity(newval)**

Modifie la luminosité des leds informatives du module.

**module→set\_usbBandwidth(newval)**

Modifie le nombre d'interface USB utilisé par le module.

**module→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**module→triggerFirmwareUpdate(secBeforeReboot)**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

**module→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YModule.FindModule()****YModule****yFindModule()yFindModule( )**

Permet de retrouver un module d'après son numéro de série ou son nom logique.

**YModule\* yFindModule( string 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()**

**YModule**

### **yFirstModule()yFirstModule( )**

Commence l'énumération des modules accessibles par la librairie.

**YModule\* yFirstModule( )**

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()****YModule**

Retourne un court texte décrivant le module.

```
string describe( )
```

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

**Retourne :**

une chaîne de caractères décrivant le module

## module→download()module→download()

YModule

Télécharge le fichier choisi du module et retourne son contenu.

```
string download( string pathname)
```

**Paramètres :**

**pathname** nom complet du fichier

**Retourne :**

le contenu du fichier chargé

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

---

<b>module</b> → <b>functionCount()</b> <b>module</b> → <b>functionCount( )</b>	<b>YModule</b>
-----------------------------------------------------------------------------------	----------------

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

**int functionCount( )**

**Retourne :**

le nombre de fonctions sur le module

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→functionId()****module→functionId( )**

**YModule**

Retourne l'identifiant matériel de la *n*ième fonction du module.

string **functionId( int 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()****YModule**

Retourne le nom logique de la *n*ième fonction du module.

```
string functionName( int functionIndex)
```

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant au nom logique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module→functionValue()** **module→**  
**functionValue( )**

---

**YModule**

Retourne la valeur publiée par la *n*ième fonction du module.

**string functionValue( int 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.

---

<b>module-&gt;get_beacon()</b>	<b>YModule</b>
<b>module-&gt;beacon()module-&gt;get_beacon( )</b>	

---

Retourne l'état de la balise de localisation.

**Y\_BEACON\_enum 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()**  
**module→errorMessage()** module→  
**get\_errorMessage( )**

---

**YModule**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

---

**module->get\_errorType()****YModule****module->errorType() module->get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

**module→get\_firmwareRelease()**  
**module→firmwareRelease()module→**  
**get\_firmwareRelease( )**

---

**YModule**

Retourne la version du logiciel embarqué du module.

**string 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.

---

<b>module→get_hardwareId()</b>	<b>YModule</b>
<b>module→hardwareId()</b>	<b>module→get_hardwareId()</b>

---

Retourne l'identifiant unique du module.

string **get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

**Retourne :**

une chaîne de caractères identifiant la fonction

**module→get\_icon2d()**  
**module→icon2d()module→get\_icon2d( )**

---

**YModule**

Retourne l'icône du module.

**string get\_icon2d( )**

L'icone est au format PNG et a une taille maximale de 1536 octets.

**Retourne :**

un buffer binaire contenant l'icone, au format png.

---

<b>module→get_lastLogs()</b>	<b>YModule</b>
<b>module→lastLogs()module→get_lastLogs( )</b>	

---

Retourne une chaîne de caractère contenant les derniers logs du module.

string **get\_lastLogs( )**

Cette méthode retourne les derniers logs qui sont encore stocké dans le module.

**Retourne :**

une chaîne de caractère contenant les derniers logs du module.

**module→get\_logicalName()** **YModule**  
**module→logicalName()** **module→get\_logicalName( )**

---

Retourne le nom logique du module.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**module→get\_luminosity()****YModule****module→luminosity()module→get\_luminosity()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
int 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.

---

<b>module→get_persistentSettings()</b>	<b>YModule</b>
<b>module→persistentSettings()module→</b>	
<b>get_persistentSettings( )</b>	

---

Retourne l'état courant des réglages persistents du module.

**Y\_PERSISTENTSETTINGS\_enum get\_persistentSettings( )**

**Retourne :**

une valeur parmi Y\_PERSISTENTSETTINGS\_LOADED, Y\_PERSISTENTSETTINGS\_SAVED et Y\_PERSISTENTSETTINGS\_MODIFIED représentant l'état courant des réglages persistents du module

En cas d'erreur, déclenche une exception ou retourne Y\_PERSISTENTSETTINGS\_INVALID.

**module→get\_productId()****YModule****module→productId()module→get\_productId()**

Retourne l'identifiant USB du module, préprogrammé en usine.

```
int get_productId( )
```

**Retourne :**

un entier représentant l'identifiant USB du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTID\_INVALID.

**module→get\_productName()**  
**module→productName()** module→  
**get\_productName( )**

---

**YModule**

Retourne le nom commercial du module, préprogrammé en usine.

string **get\_productName( )**

**Retourne :**

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTNAME\_INVALID.

---

```
module->get_productRelease()  
module->productRelease()module->  
get_productRelease( )
```

**YModule**

Retourne le numéro de version matériel du module, préprogrammé en usine.

```
int get_productRelease( )
```

**Retourne :**

un entier représentant le numéro de version matériel du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTRELEASE\_INVALID.

**module→get\_rebootCountdown()** **YModule**  
**module→rebootCountdown()****module→get\_rebootCountdown( )**

---

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

**int get\_rebootCountdown( )**

**Retourne :**

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

En cas d'erreur, déclenche une exception ou retourne Y\_REBOOTCOUNTDOWN\_INVALID.

**module→get\_serialNumber()**  
**module→serialNumber()** module→  
**get\_serialNumber( )**

**YModule**

Retourne le numéro de série du module, préprogrammé en usine.

string **get\_serialNumber( )**

**Retourne :**

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_SERIALNUMBER\_INVALID.

**module→get\_upTime()**  
**module→upTime()module→get\_upTime( )**

---

**YModule**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

s64 **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.

---

<b>module-&gt;get_usbBandwidth()</b>	<b>YModule</b>
<b>module-&gt;usbBandwidth()</b> <b>module-&gt;</b>	
<b>get_usbBandwidth()</b>	

---

Retourne le nombre d'interface USB utilisé par le module.

**Y\_USBBANDWIDTH\_enum get\_usbBandwidth( )**

**Retourne :**

soit Y\_USBBANDWIDTH\_SIMPLE, soit Y\_USBBANDWIDTH\_DOUBLE, selon le nombre d'interface USB utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_USBBANDWIDTH\_INVALID.

**module→get\_usbCurrent()**

**YModule**

**module→usbCurrent()module→get\_usbCurrent( )**

---

Retourne le courant consommé par le module sur le bus USB, en milliampères.

**int get\_usbCurrent( )**

**Retourne :**

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

En cas d'erreur, déclenche une exception ou retourne Y\_USBCURRENT\_INVALID.

**module→get(userData)****YModule****module→userData()module→get(userData( )**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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.

**module->isOnline()****YModule**

Vérifie si le module est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le module est joignable, false sinon

**module→load()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module->nextModule()**

**YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

`YModule * nextModule()`

**Retourne :**

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**module→reboot()****YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
int reboot( int secBeforeReboot)
```

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→registerLogCallback()**  
**module→registerLogCallback( )**

---

**YModule**

todo

**void registerLogCallback( YModuleLogCallback callback)**

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

<b>module→revertFromFlash()</b>	<b>module→revertFromFlash( )</b>	<b>YModule</b>
---------------------------------	----------------------------------	----------------

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

```
int 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()**

**YModule**

Sauve les réglages courants dans la mémoire non volatile du module.

```
int saveToFlash( )
```

Attention le nombre total de sauvegardes possibles durant la vie du module est limité (environ 100000 cycles). Nappelez 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.

---

<b>module-&gt;set_beacon()</b>	<b>YModule</b>
<b>module-&gt;setBeacon()</b>	

---

Allume ou éteint la balise de localisation du module.

```
int set_beacon( Y_BEACON_enum 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()**  
**module->setLogicalName()** **module->**  
**set\_logicalName( )**

**YModule**

Change le nom logique du module.

**int set\_logicalName( const string& 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.

---

<b>module→set_luminosity()</b> <b>module→setLuminosity()</b> <b>module→</b> <b>set_luminosity( )</b>	<b>YModule</b>
------------------------------------------------------------------------------------------------------------	----------------

---

Modifie la luminosité des leds informatives du module.

**int set\_luminosity( int newval)**

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la luminosité des leds informatives du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

```
module->set_usbBandwidth()  
module->setUsbBandwidth()module->  
set_usbBandwidth( )
```

**YModule**

Modifie le nombre d'interface USB utilisé par le module.

```
int set_usbBandwidth( Y_USBWIDTH_enum newval)
```

Vous devez redémarrer le module après avoir changé ce réglage.

**Paramètres :**

**newval** soit Y\_USBWIDTH\_SIMPLE, soit Y\_USBWIDTH\_DOUBLE, selon le nombre d'interface USB utilisé par le module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→set(userData)****YModule****module→setUserData() module→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**module→triggerFirmwareUpdate()**  
**module→triggerFirmwareUpdate( )**

**YModule**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

**int triggerFirmwareUpdate( int secBeforeReboot)**

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.26. Interface de la fonction Network

Les objets YNetwork permettent de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_network.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YNetwork = yoctolib.YNetwork;
php	require_once('yocto_network.php');
cpp	#include "yocto_network.h"
m	#import "yocto_network.h"
pas	uses yocto_network;
vb	yocto_network.vb
cs	yocto_network.cs
java	import com.yoctopuce.YoctoAPI.YNetwork;
py	from yocto_network import *

### 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 laissez-passer pour s'y connecter.

#### network→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format TYPE ( NAME )=SERIAL.FUNCTIONID.

#### network→get\_adminPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

#### network→get\_advertisedValue()

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

#### network→get\_callbackCredentials()

Retourne une version hashée du laissez-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

#### network→get\_callbackEncoding()

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

#### network→get\_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

#### network→get\_callbackMethod()

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

#### network→get\_callbackMinDelay()

Retourne l'attente minimale entre deux notifications par callback, en secondes.

#### network→get\_callbackUrl()

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

#### network→get\_discoverable()

### 3. Reference

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

#### **network→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_friendlyName()**

Retourne un identifiant global de l'interface réseau au format NOM\_MODULE . NOM\_FONCTION.

#### **network→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **network→get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

#### **network→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL . FUNCTIONID.

#### **network→get\_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

#### **network→get\_logicalName()**

Retourne le nom logique de l'interface réseau.

#### **network→get\_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

#### **network→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

#### **network→get\_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

#### **network→get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

#### **network→get\_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

#### **network→get\_secondaryDNS()**

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

#### **network→get\_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

#### **network→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **network→get\_userPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

#### **network→get\_wwwWatchdogDelay()**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

#### **network→isOnline()**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

**network→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

**network→load(msValidity)**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

**network→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

**network→nextNetwork()**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

**network→ping(host)**

Ping `str_host` pour vérifier la connexion réseau.

**network→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**network→set\_adminPassword(newval)**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

**network→set\_callbackCredentials(newval)**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

**network→set\_callbackEncoding(newval)**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

**network→set\_callbackMaxDelay(newval)**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

**network→set\_callbackMethod(newval)**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

**network→set\_callbackMinDelay(newval)**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

**network→set\_callbackUrl(newval)**

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

**network→set\_discoverable(newval)**

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

**network→set\_logicalName(newval)**

Modifie le nom logique de l'interface réseau.

**network→set\_primaryDNS(newval)**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

**network→set\_secondaryDNS(newval)**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

**network→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**network→set\_userPassword(newval)**

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

**network→set\_wwwWatchdogDelay(newval)**

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

**network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)**

### **3. Reference**

---

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

**network→useStaticIP(ipAddress, subnetMaskLen, router)**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

**network→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YNetwork.FindNetwork()****YNetwork****yFindNetwork()yFindNetwork( )**

Permet de retrouver une interface réseau d'après un identifiant donné.

**YNetwork\* yFindNetwork( const string& 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()**

**YNetwork**

### **yFirstNetwork()yFirstNetwork( )**

Commence l'énumération des interfaces réseau accessibles par la librairie.

**YNetwork\* yFirstNetwork( )**

Utiliser la fonction `YNetwork.nextNetwork()` pour itérer sur les autres interfaces réseau.

**Retourne :**

un pointeur sur un objet `YNetwork`, correspondant à la première interface réseau accessible en ligne, ou `null` si il n'y a pas de interfaces réseau disponibles.

**network→callbackLogin()network→  
callbackLogin( )****YNetwork**

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

```
int callbackLogin( string username, string 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()****YNetwork**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format TYPE ( NAME )=SERIAL.FUNCTIONID.

string **describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant l'interface réseau (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**network→get\_adminPassword()**  
**network→adminPassword()network→**  
**get\_adminPassword( )**

**YNetwork**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

**string 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()**  
**network→advertisedValue()network→**  
**get\_advertisedValue( )**

---

**YNetwork**

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'interface réseau (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**network→get\_callbackCredentials()**

**YNetwork**

**network→callbackCredentials()network→  
get\_callbackCredentials()**

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

**string get\_callbackCredentials( )**

**Retourne :**

une chaîne de caractères représentant une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKCREDENTIALS\_INVALID.

**network→get\_callbackEncoding()**  
**network→callbackEncoding()network→**  
**get\_callbackEncoding( )**

**YNetwork**

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

[Y\\_CALLBACKENCODING\\_enum get\\_callbackEncoding\( \)](#)

**Retourne :**

une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKENCODING\_INVALID.

**network→get\_callbackMaxDelay()**  
**network→callbackMaxDelay()network→**  
**get\_callbackMaxDelay( )**

**YNetwork**

Retourne l'attente maximale entre deux notifications par callback, en secondes.

```
int get_callbackMaxDelay( )
```

**Retourne :**

un entier représentant l'attente maximale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMAXDELAY\_INVALID.

**network→get\_callbackMethod()**  
**network→callbackMethod()network→**  
**get\_callbackMethod( )**

**YNetwork**

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

[Y\\_CALLBACKMETHOD\\_enum get\\_callbackMethod\( \)](#)

**Retourne :**

une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMETHOD\_INVALID.

---

<b>network→get_callbackMinDelay()</b>	<b>YNetwork</b>
<b>network→callbackMinDelay()network→get_callbackMinDelay( )</b>	

---

Retourne l'attente minimale entre deux notifications par callback, en secondes.

```
int 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()**  
**network→callbackUrl()network→**  
**get\_callbackUrl( )**

**YNetwork**

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

string **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**.

---

<b>network→get_discoverable()</b>	<b>YNetwork</b>
<b>network→discoverable()network→get_discoverable()</b>	

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

**Y\_DISCOVERABLE\_enum get\_Discoverable( )**

**Retourne :**

soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne Y\_DISCOVERABLE\_INVALID.

**network→get\_errorMessage()**  
**network→errorMessage()network→**  
**get\_errorMessage( )**

---

**YNetwork**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

**network→get\_errorType()****YNetwork****network→errorType()network→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

**network→get\_friendlyName()** YNetwork

**network→friendlyName()network→get\_friendlyName( )**

---

Retourne un identifiant global de l'interface réseau au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et de l'interface réseau si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'interface réseau en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**network→get\_functionDescriptor()**  
**network→functionDescriptor()network→**  
**get\_functionDescriptor( )**

**YNetwork**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**network→get\_functionId()**

**YNetwork**

**network→functionId()network→get\_functionId( )**

---

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

string **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

<b>network→get_hardwareId()</b>	<b>YNetwork</b>
<b>network→hardwareId()</b>	<b>network→</b>
<b>get_hardwareId( )</b>	

---

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**network→get\_ipAddress()**

**YNetwork**

**network→ipAddress()network→get\_ipAddress( )**

---

Retourne l'adresse IP utilisée par le module Yoctopuce.

**string 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()**  
**network→logicalName()network→**  
**get\_logicalName( )**

**YNetwork**

Retourne le nom logique de l'interface réseau.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**network→get\_macAddress()**  
**network→macAddress()network→**  
**get\_macAddress( )**

---

**YNetwork**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

**string get\_macAddress( )**

L'adresse MAC est aussi présente sur un autocollant sur le module, représentée en chiffres et en code-barres.

**Retourne :**

une chaîne de caractères représentant l'adresse MAC de l'interface réseau, unique pour chaque module

En cas d'erreur, déclenche une exception ou retourne **Y\_MACADDRESS\_INVALID**.

**network→get\_module()****YNetwork****network→module()network→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**une instance de `YModule`

**network→get\_poeCurrent()** YNetwork  
**network→poeCurrent()network→get\_poeCurrent( )**

---

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

**int get\_poeCurrent( )**

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

**Retourne :**

un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne Y\_POECURRENT\_INVALID.

**network→get\_primaryDNS()**  
**network→primaryDNS()network→**  
**get\_primaryDNS( )**

**YNetwork**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

**string get\_primaryDNS( )**

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_PRIMARYDNS\_INVALID.

**network→get\_readiness()****YNetwork****network→readiness()network→get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

**Y\_READINESS\_enum get\_readiness( )**

Le niveau zéro (DOWN\_0) signifie qu'aucun support réseau matériel n'a été détecté. Soit il n'y a pas de signal sur le câble réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE\_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme la 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 sécurités 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 une serveur NTP.

**Retourne :**

une valeur parmi Y\_READINESS\_DOWN, Y\_READINESS\_EXISTS, Y\_READINESS\_LINKED, Y\_READINESS\_LAN\_OK et Y\_READINESS\_WWW\_OK représentant l'état de fonctionnement atteint par l'interface réseau

En cas d'erreur, déclenche une exception ou retourne Y\_READINESS\_INVALID.

---

<b>network→get_router()</b>	<b>YNetwork</b>
<b>network→router()network→get_router( )</b>	

---

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

**string 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()**  
**network→secondaryDNS()network→**  
**get\_secondaryDNS( )**

---

**YNetwork**

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

**string get\_secondaryDNS( )**

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_SECONDARYDNS\_INVALID.

**network→get\_subnetMask()**  
**network→subnetMask()network→**  
**get\_subnetMask( )**

**YNetwork**

Retourne le masque de sous-réseau utilisé par le module.

**string get\_subnetMask( )**

**Retourne :**

une chaîne de caractères représentant le masque de sous-réseau utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_SUBNETMASK\_INVALID.

**network→get(userData)**

**YNetwork**

**network→userData()network→get(userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * get(userData( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**network→get\_userPassword()**  
**network→userPassword()network→**  
**get\_userPassword( )**

**YNetwork**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

```
string get_userPassword( )
```

**Retourne :**

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_USERPASSWORD\_INVALID.

---

<b>network→get_wwwWatchdogDelay()</b>	<b>YNetwork</b>
<b>network→wwwWatchdogDelay()network→</b>	
<b>get_wwwWatchdogDelay( )</b>	

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

```
int get_wwwWatchdogDelay( )
```

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW.

**Retourne :**

un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

En cas d'erreur, déclenche une exception ou retourne Y\_WWWWATCHDOGDELAY\_INVALID.

**network→isOnline()****YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache de l'interface réseau sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'interface réseau est joignable, false sinon

**network→load()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→nextNetwork()network→nextNetwork( )****YNetwork**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

`YNetwork * nextNetwork( )`

**Retourne :**

un pointeur sur un objet `YNetwork` accessible en ligne, ou `null` lorsque l'énumération est terminée.

## network→ping() network→ping()

YNetwork

Ping str\_host pour vérifier la connexion réseau.

**string ping( string host)**

Envoie quatre requêtes ICMP ECHO\_RESPONER à la cible str\_host depuis le module. Cette méthode retourne une chaîne de caractères avec le résultat des 4 requêtes ICMP ECHO\_RESPONSE.

**Paramètres :**

**host** le nom d'hôte ou l'adresse IP de la cible

**Retourne :**

une chaîne de caractères contenant le résultat du ping.

**network→registerValueCallback()network→registerValueCallback( )****YNetwork**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YNetworkValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**network→set\_adminPassword()** YNetwork

**network→setAdminPassword()network→set\_adminPassword( )**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

**int set\_adminPassword( const string& 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.

---

<b>network→set_callbackCredentials()</b>	<b>YNetwork</b>
<b>network→setCallbackCredentials()network→</b> <b>set_callbackCredentials()</b>	

Modifie le laisser-passer pour se connecter à l'adresse de callback.

```
int set_callbackCredentials( const string& newval)
```

Le laisser-passer doit être fourni tel que retourné par la fonction `get_callbackCredentials`, sous la forme `username:hash`. La valeur du hash dépend de la méthode d'autorisation implémentée par le callback. Pour une autorisation de type Basic, le hash est le MD5 de la chaîne `username:password`. Pour une autorisation de type Digest, le hash est le MD5 de la chaîne `username:realm:password`. Pour une utilisation simplifiée, utilisez la fonction `callbackLogin`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le laisser-passer pour se connecter à l'adresse de callback

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackEncoding()** **YNetwork**  
**network→setCallbackEncoding()** **network→set\_callbackEncoding( )**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
int set_callbackEncoding( Y_CALLBACKENCODING_enum newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMaxDelay()****YNetwork****network→setCallbackMaxDelay()network→  
set\_callbackMaxDelay( )**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

**int set\_callbackMaxDelay( int newval)****Paramètres :**

**newval** un entier représentant l'attente maximale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMethod()**  
**network→setCallbackMethod()** **network→set\_callbackMethod( )**

**YNetwork**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
int set_callbackMethod( Y_CALLBACKMETHOD_enum newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**network→set\_callbackMinDelay()** YNetwork

**network→setCallbackMinDelay()network→set\_callbackMinDelay( )**

---

Modifie l'attente minimale entre deux notifications par callback, en secondes.

**int set\_callbackMinDelay( int 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()**  
**network→setCallbackUrl()** **network→set\_callbackUrl( )**

**YNetwork**

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

**int set\_callbackUrl( const string& 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.

---

<b>network→set_discoverable()</b>	<b>YNetwork</b>
<b>network→setDiscoverable()network→set_discoverable()</b>	

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

```
int set_discoverable( Y_DISCOVERABLE_enum newval)
```

**Paramètres :**

**newval** soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_logicalName()**  
**network→setLogicalName()** **network→set\_logicalName( )**

**YNetwork**

Modifie le nom logique de l'interface réseau.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_primaryDNS()**  
**network→setPrimaryDNS()network→**  
**set\_primaryDNS( )**

**YNetwork**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

**int set\_primaryDNS( const string& newval)**

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_secondaryDNS()**  
**network→setSecondaryDNS()** **network→set\_secondaryDNS( )**

**YNetwork**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

**int set\_secondaryDNS( const string& newval)**

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set(userData())****YNetwork****network→setUserData()network→set(userData())**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**network→set\_userPassword()**  
**network→setUserPassword()network→**  
**set\_userPassword( )**

**YNetwork**

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

**int set\_userPassword( const string& 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.

---

<b>network→set_wwwWatchdogDelay()</b>	<b>YNetwork</b>
<b>network→setWwwWatchdogDelay()network→</b>	
<b>set_wwwWatchdogDelay( )</b>	

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

**int set\_wwwWatchdogDelay( int newval)**

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW. La plus petite durée non-nulle utilisable est 90 secondes.

**Paramètres :**

**newval** un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useDHCP()network→useDHCP( )****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

```
int useDHCP( string fallbackIpAddr,  
              int fallbackSubnetMaskLen,  
              string fallbackRouter)
```

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode saveToFlash( ) et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

<b>fallbackIpAddr</b>	adresse IP à utiliser si aucun serveur DHCP ne répond
<b>fallbackSubnetMaskLen</b>	longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.
<b>fallbackRouter</b>	adresse de la passerelle à utiliser si aucun serveur DHCP ne répond

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useStaticIP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

```
int useStaticIP( string ipAddress,  
                  int subnetMaskLen,  
                  string 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.

## 3.27. contrôle d'OS

L'objet OsControl permet de contrôler le système d'exploitation sur lequel tourne un VirtualHub. OsControl n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activé au lancement du VirtualHub, avec l'option -o.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_oscontrol.js'></script>
nodejs var yoctolib = require('yoctolib');
var YOsControl = yoctolib.YOsControl;
require_once('yocto_oscontrol.php');
cpp #include "yocto_oscontrol.h"
m #import "yocto_oscontrol.h"
pas uses yocto_oscontrol;
vb yocto_oscontrol.vb
cs yocto_oscontrol.cs
java import com.yoctopuce.YoctoAPI.YOsControl;
py from yocto_oscontrol import *

```

### Fonction globales

#### **yFindOsControl(func)**

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

#### **yFirstOsControl()**

Commence l'énumération des contrôle d'OS accessibles par la librairie.

### Méthodes des objets YOsControl

#### **oscontrol→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format TYPE (NAME )=SERIAL . FUNCTIONID.

#### **oscontrol→get\_advertisedValue()**

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

#### **oscontrol→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

#### **oscontrol→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

#### **oscontrol→get\_friendlyName()**

Retourne un identifiant global du contrôle d'OS au format NOM\_MODULE . NOM\_FONCTION.

#### **oscontrol→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **oscontrol→get\_functionId()**

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

#### **oscontrol→get\_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'OS au format SERIAL . FUNCTIONID.

#### **oscontrol→get\_logicalName()**

Retourne le nom logique du contrôle d'OS.

#### **oscontrol→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **oscontrol→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**oscontrol->get\_shutdownCountdown()**

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

**oscontrol->get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**oscontrol->isOnline()**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**oscontrol->isOnline\_async(callback, context)**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**oscontrol->load(msValidity)**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**oscontrol->load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**oscontrol->nextOsControl()**

Continue l'énumération des contrôle d'OS commencée à l'aide de yFirstOsControl( ).

**oscontrol->registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**oscontrol->set\_logicalName(newval)**

Modifie le nom logique du contrôle d'OS.

**oscontrol->set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**oscontrol->shutdown(secBeforeShutDown)**

Agende un arrêt de l'OS dans un nombre donné de secondes.

**oscontrol->wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YOsControl.FindOsControl() yFindOsControl()yFindOsControl( )

YOsControl

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

YOsControl\* **yFindOsControl( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'OS soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode YOsControl.isOnLine( ) pour tester si le contrôle d'OS est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

### Retourne :

un objet de classe YOsControl qui permet ensuite de contrôler le contrôle d'OS.

**YOsControl.FirstOsControl()****YOsControl****yFirstOsControl()yFirstOsControl( )**

Commence l'énumération des contrôle d'OS accessibles par la librairie.

**YOsControl\* yFirstOsControl( )**

Utiliser la fonction YOsControl.nextOsControl( ) pour itérer sur les autres contrôle d'OS.

**Retourne :**

un pointeur sur un objet YOsControl, correspondant au premier contrôle d'OS accessible en ligne, ou null si il n'y a pas de contrôle d'OS disponibles.

**oscontrol→describe()****YOsControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le contrôle d'OS (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**oscontrol→get\_advertisedValue()****YOsControl****oscontrol→advertisedValue()oscontrol→  
get\_advertisedValue( )**

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

```
string get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**oscontrol→get\_errorMessage()**

YOsControl

**oscontrol→errorMessage()oscontrol→  
get\_errorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

---

**oscontrol→get\_errorType()**  
**oscontrol→errorType()**oscontrol→  
**get\_errorType( )**

**YOsControl**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

---

<b>oscontrol→get_friendlyName()</b>	<b>YOsControl</b>
<b>oscontrol→friendlyName()</b> oscontrol→ <b>get_friendlyName( )</b>	

---

Retourne un identifiant global du contrôle d'OS au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du contrôle d'OS si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'OS (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

---

<b>oscontrol→get_functionDescriptor()</b>	<b>YOsControl</b>
<b>oscontrol→functionDescriptor()oscontrol→get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>oscontrol→get_functionId()</b>	<b>YOsControl</b>
<b>oscontrol→functionId()oscontrol→get_functionId( )</b>	

---

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

---

<b>oscontrol→get_hardwareId()</b>	<b>YOsControl</b>
<b>oscontrol→hardwareId()</b> oscontrol→ <b>get_hardwareId( )</b>	

---

Retourne l'identifiant matériel unique du contrôle d'OS au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'OS (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**oscontrol→get\_logicalName()**  
**oscontrol→logicalName()****oscontrol→get\_logicalName( )**

---

**YOsControl**

Retourne le nom logique du contrôle d'OS.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du contrôle d'OS. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**oscontrol→get\_module()****YOsControl****oscontrol→module()oscontrol→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
YModule * get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

<b>oscontrol→get_shutdownCountdown()</b>	<b>YOsControl</b>
<b>oscontrol→shutdownCountdown()</b>	<b>oscontrol→get_shutdownCountdown( )</b>

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

```
int get_shutdownCountdown( )
```

**Retourne :**

un entier représentant le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé

En cas d'erreur, déclenche une exception ou retourne Y\_SHUTDOWNCOUNTDOWN\_INVALID.

**oscontrol→get(userData)****YOsControl****oscontrol→userData()oscontrol→get(userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**oscontrol→isOnline()****YOsControl**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le contrôle d'OS est joignable, false sinon

**oscontrol→load()****YOscControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**oscontrol→nextOsControl()**  
**oscontrol→nextOsControl( )**

**YOsControl**

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

**YOsControl \* nextOsControl( )**

**Retourne :**

un pointeur sur un objet `YOsControl` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**oscontrol→registerValueCallback()**  
**oscontrol→registerValueCallback( )****YOsControl**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YOsControlValueCallback 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 appellés 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.

---

<b>oscontrol→set_logicalName()</b>	<b>YOsControl</b>
<b>oscontrol→setLogicalName()</b> oscontrol→ <b>set_logicalName( )</b>	

---

Modifie le nom logique du contrôle d'OS.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'OS.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**oscontrol→set(userData)**  
**oscontrol→setUserData()****oscontrol→set(userData)**

**YOscControl**

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)`.

`void set(userData( void* data)`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

`data` objet quelconque à mémoriser

## **oscontrol→shutdown()**

**YOsControl**

---

Agende un arrêt de l'OS dans un nombre donné de secondes.

```
int shutdown( int secBeforeShutDown)
```

**Paramètres :**

**secBeforeShutDown** nombre de secondes avant l'arrêt

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.28. Interface de la fonction Power

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_power.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YPower = yoctolib.YPower;
php	require_once('yocto_power.php');
cpp	#include "yocto_power.h"
m	#import "yocto_power.h"
pas	uses yocto_power;
vb	yocto_power.vb
cs	yocto_power.cs
java	import com.yoctopuce.YoctoAPI.YPower;
py	from yocto_power import *

### Fonction globales

#### yFindPower(func)

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

#### yFirstPower()

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

### Méthodes des objets YPower

#### power→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### power→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### power→get\_advertisedValue()

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

#### power→get\_cosPhi()

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

#### power→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### power→get\_currentValue()

Retourne la valeur instantanée de la puissance électrique.

#### power→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

#### power→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

#### power→get\_friendlyName()

Retourne un identifiant global du capteur de puissance électrique au format NOM\_MODULE . NOM\_FONCTION.

#### power→get\_functionDescriptor()

### 3. Reference

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **power→get\_functionId()**

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

#### **power→get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

#### **power→get\_highestValue()**

Retourne la valeur maximale observée pour la puissance électrique.

#### **power→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

#### **power→get\_logicalName()**

Retourne le nom logique du capteur de puissance électrique.

#### **power→get\_lowestValue()**

Retourne la valeur minimale observée pour la puissance électrique.

#### **power→get\_meter()**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

#### **power→get\_meterTimer()**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

#### **power→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **power→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **power→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

#### **power→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

#### **power→get\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **power→get\_unit()**

Retourne l'unité dans laquelle la puissance électrique est exprimée.

#### **power→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **power→isOnline()**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

#### **power→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

#### **power→load(msValidity)**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

#### **power→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **power→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

**power→nextPower()**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

**power→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**power→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**power→reset()**

Réinitialise le compteur d'énergie.

**power→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

**power→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**power→set\_logicalName(newval)**

Modifie le nom logique du capteur de puissance électrique.

**power→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la puissance électrique.

**power→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**power→set\_resolution(newval)**

Modifie la résolution des valeurs mesurées.

**power→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**power→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPower.FindPower()****YPower****yFindPower()yFindPower( )**

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

**YPower\* yFindPower( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de puissance électrique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance électrique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de puissance électrique sans ambiguïté

**Retourne :**

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance électrique.

**YPower.FirstPower()****YPower****yFirstPower()yFirstPower( )**

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

**YPower\* yFirstPower( )**

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance électrique.

**Retourne :**

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance électrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance électrique disponibles.

**power→calibrateFromPoints()power→  
calibrateFromPoints( )****YPower**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→describe()****YPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format TYPE ( NAME ) =SERIAL . FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de puissance électrique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**power→get\_advertisedValue()** YPower  
**power→advertisedValue()** power→  
**get\_advertisedValue( )**

---

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

string **get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de puissance électrique (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**power→get\_cosPhi()****YPower****power→cosPhi()power→get\_cosPhi( )**

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

```
double get_cosPhi( )
```

**Retourne :**

une valeur numérique représentant le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA)

En cas d'erreur, déclenche une exception ou retourne Y\_COSPHI\_INVALID.

**power→get\_currentRawValue()** YPower  
**power→currentRawValue()power→**  
**get\_currentRawValue( )**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

double **get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**power→get\_currentValue()**  
**power→currentValue()** power→  
**get\_currentValue()**

**YPower**

Retourne la valeur instantanée de la puissance électrique.

double **get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur instantanée de la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**power→getErrorMessage()**  
**power→errorMessage()****power→getErrorMessage( )**

**YPower**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

**power→get\_errorType()****YPower****power→errorType()power→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

**power→get\_friendlyName()**  
**power→friendlyName()** power→  
**get\_friendlyName( )**

**YPower**

Retourne un identifiant global du capteur de puissance électrique au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de puissance électrique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de puissance électrique (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**power→get\_functionDescriptor()**  
**power→functionDescriptor()** power→  
**get\_functionDescriptor( )**

**YPower**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**power→get\_functionId()**

**YPower**

**power→functionId()power→get\_functionId()**

---

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

string **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**power→get\_hardwareId()****YPower****power→hardwareId()power→get\_hardwareId( )**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de puissance électrique (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**power→get\_highestValue()** YPower  
**power→highestValue()power→**  
**get\_highestValue( )**

---

Retourne la valeur maximale observée pour la puissance électrique.

double **get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

---

<b>power→get_logFrequency()</b>	<b>YPower</b>
<b>power→logFrequency()</b>	<b>power→</b>
<b>get_logFrequency( )</b>	

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
string get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**power→get\_logicalName()**

**YPower**

**power→logicalName()power→get\_logicalName( )**

---

Retourne le nom logique du capteur de puissance électrique.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de puissance électrique. En cas d'erreur, déclenche une exception ou retourne **Y\_LOGICALNAME\_INVALID**.

**power→get\_lowestValue()****YPower****power→lowestValue()power→get\_lowestValue()**

Retourne la valeur minimale observée pour la puissance électrique.

```
double get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**power→get\_meter()****YPower****power→meter()power→get\_meter( )**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

```
double get_meter()
```

Ce compteur est réinitialisé à chaque démarrage du module.

**Retourne :**

une valeur numérique représentant la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée

En cas d'erreur, déclenche une exception ou retourne Y\_METER\_INVALID.

**power→get\_meterTimer()****YPower****power→meterTimer()power→get\_meterTimer( )**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

**int get\_meterTimer( )****Retourne :**

un entier représentant le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_METERTIMER\_INVALID.

**power→get\_module()**  
**power→module()power→get\_module()**

---

**YPower**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**power→get\_recordedData()**  
**power→recordedData()power→**  
**get\_recordedData()**

**YPower**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**power→get\_reportFrequency()** YPower  
**power→reportFrequency()** power→  
**get\_reportFrequency( )**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**power→get\_resolution()****YPower****power→resolution()power→get\_resolution( )**

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**power→get\_unit()**

**YPower**

**power→unit()power→get\_unit()**

---

Retourne l'unité dans laquelle la puissance électrique est exprimée.

string **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la puissance électrique est exprimée

En cas d'erreur, déclenche une exception ou retourne **Y\_UNIT\_INVALID**.

**power→get(userData)****YPower****power→userData() power→get(userData)()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**power→isOnline()****YPower**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du capteur de puissance électrique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de puissance électrique est joignable, false sinon

**power→load()****YPower**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→loadCalibrationPoints()power→  
loadCalibrationPoints()****YPower**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→nextPower()****YPower**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

`YPower * nextPower()`

**Retourne :**

un pointeur sur un objet `YPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

```
power->registerTimedReportCallback() power->  
registerTimedReportCallback()
```

YPower

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YPowerTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**power→registerValueCallback()**  
**power→registerValueCallback( )****YPower**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YPowerValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## power→reset() power→reset( )

YPower

Réinitialise le compteur d'énergie.

```
int reset( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_highestValue()**  
**power→setHighestValue()** power→  
**set\_highestValue()**

**YPower**

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la puissance électrique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_logFrequency()**  
**power→setLogFrequency()** power→  
**set\_logFrequency( )**

**YPower**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_logicalName()**  
**power→setLogicalName()** power→  
**set\_logicalName( )**

**YPower**

Modifie le nom logique du capteur de puissance électrique.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de puissance électrique.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_lowestValue()** YPower  
**power→setLowestValue()** power→  
**set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour la puissance électrique.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la puissance électrique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_reportFrequency()**  
**power→setReportFrequency()power→**  
**set\_reportFrequency( )**

**YPower**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_resolution()** YPower  
**power→setResolution()****power→set\_resolution( )**

---

Modifie la résolution des valeurs mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set(userData)****YPower****power→setUserData()power→set(userData( )**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.29. Interface de la fonction Pressure

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_pressure.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPressure = yoctolib.YPressure;
php require_once('yocto_pressure.php');
cpp #include "yocto_pressure.h"
m #import "yocto_pressure.h"
pas uses yocto_pressure;
vb yocto_pressure.vb
cs yocto_pressure.cs
java import com.yoctopuce.YoctoAPI.YPressure;
py from yocto_pressure import *

```

### Fonction globales

#### yFindPressure(func)

Permet de retrouver un capteur de pression d'après un identifiant donné.

#### yFirstPressure()

Commence l'énumération des capteurs de pression accessibles par la librairie.

### Méthodes des objets YPressure

#### pressure→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### pressure→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### pressure→get\_advertisedValue()

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

#### pressure→get\_currentRawValue()

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

#### pressure→get\_currentValue()

Retourne la mesure actuelle de la pression.

#### pressure→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### pressure→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### pressure→get\_friendlyName()

Retourne un identifiant global du capteur de pression au format NOM\_MODULE . NOM\_FONCTION.

#### pressure→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### pressure→get\_functionId()

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

#### pressure→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de pression au format SERIAL.FUNCTIONID.
<b>pressure→get_highestValue()</b>
Retourne la valeur maximale observée pour la pression.
<b>pressure→get_logFrequency()</b>
Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>pressure→get_logicalName()</b>
Retourne le nom logique du capteur de pression.
<b>pressure→get_lowestValue()</b>
Retourne la valeur minimale observée pour la pression.
<b>pressure→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pressure→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pressure→get_recordedData(startTime, endTime)</b>
Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>pressure→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>pressure→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>pressure→get_unit()</b>
Retourne l'unité dans laquelle la pression est exprimée.
<b>pressure→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>pressure→isOnline()</b>
Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.
<b>pressure→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.
<b>pressure→load(msValidity)</b>
Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.
<b>pressure→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>pressure→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.
<b>pressure→nextPressure()</b>
Continue l'énumération des capteurs de pression commencée à l'aide de yFirstPressure( ).
<b>pressure→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>pressure→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>pressure→set_highestValue(newval)</b>
Modifie la mémoire de valeur maximale observée pour la pression.
<b>pressure→set_logFrequency(newval)</b>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**pressure→set\_logicalName(newval)**

Modifie le nom logique du capteur de pression.

**pressure→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la pression.

**pressure→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**pressure→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**pressure→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**pressure→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPressure.FindPressure()****YPressure****yFindPressure()yFindPressure( )**

Permet de retrouver un capteur de pression d'après un identifiant donné.

**YPressure\* yFindPressure( const string& 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()

**YPressure**

### yFirstPressure()yFirstPressure( )

Commence l'énumération des capteurs de pression accessibles par la librairie.

**YPressure\* yFirstPressure( )**

Utiliser la fonction `YPressure.nextPressure( )` pour itérer sur les autres capteurs de pression.

**Retourne :**

un pointeur sur un objet `YPressure`, correspondant au premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

**pressure→calibrateFromPoints()pressure→** **YPressure**  
**calibrateFromPoints( )**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**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()****YPressure**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le capteur de pression (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**pressure→get\_advertisedValue()**

**YPressure**

**pressure→advertisedValue()pressure→  
get\_advertisedValue( )**

---

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

**string 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()** YPressure  
**pressure→currentRawValue()pressure→**  
**get\_currentRawValue( )**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

double **get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**pressure→get\_currentValue()**  
**pressure→currentValue()** **pressure→get\_currentValue()**

**YPressure**

Retourne la mesure actuelle de la pression.

**double get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la mesure actuelle de la pression

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**pressure→getErrorMessage()**  
**pressure→errorMessage()** **pressure→getErrorMessage( )**

**YPressure**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

**pressure→get\_errorType()**

**YPressure**

**pressure→errorType()pressure→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

**pressure→get\_friendlyName()** YPressure  
**pressure→friendlyName()pressure→get\_friendlyName( )**

Retourne un identifiant global du capteur de pression au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de pression si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de pression (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de pression en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**pressure→get\_functionDescriptor()**  
**pressure→functionDescriptor()pressure→**  
**get\_functionDescriptor( )**

**YPressure**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>pressure→get_functionId()</b>	<b>YPressure</b>
<b>pressure→functionId()pressure→</b>	
<b>get_functionId( )</b>	

---

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le capteur de pression (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

**pressure→get\_hardwareId()**

**YPressure**

**pressure→hardwareId()pressure→  
get\_hardwareId( )**

Retourne l'identifiant matériel unique du capteur de pression au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de pression (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de pression (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**pressure→get\_highestValue()**  
**pressure→highestValue()pressure→**  
**get\_highestValue( )**

---

**YPressure**

Retourne la valeur maximale observée pour la pression.

double **get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la pression

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

**pressure→get\_logFrequency()**  
**pressure→logFrequency()pressure→**  
**get\_logFrequency( )**

**YPressure**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**pressure→get\_logicalName()** YPressure  
**pressure→logicalName()pressure→**  
**get\_logicalName( )**

---

Retourne le nom logique du capteur de pression.

string **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()**  
**pressure→lowestValue()pressure→**  
**get\_lowestValue( )**

**YPressure**

Retourne la valeur minimale observée pour la pression.

**double get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la pression

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**pressure→get\_module()**

**YPressure**

**pressure→module()pressure→get\_module( )**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**pressure→get\_recordedData()**  
**pressure→recordedData()pressure→**  
**get\_recordedData()**

**YPressure**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**pressure→get\_reportFrequency()** YPressure  
**pressure→reportFrequency()pressure→**  
**get\_reportFrequency( )**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**pressure→get\_resolution()**  
**pressure→resolution()pressure→**  
**get\_resolution( )**

**YPressure**

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne **Y\_RESOLUTION\_INVALID**.

**pressure→get\_unit()**

**YPressure**

**pressure→unit()pressure→get\_unit()**

---

Retourne l'unité dans laquelle la pression est exprimée.

**string get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la pression est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**pressure→get(userData)****YPressure****pressure→userData()pressure→get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**pressure→isOnline()****YPressure**

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du capteur de pression sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de pression est joignable, false sinon

**pressure→load()****YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure**→**loadCalibrationPoints()****pressure**→  
**loadCalibrationPoints()**

**YPressure**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→nextPressure()pressure→**  
**nextPressure( )**

**YPressure**

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

`YPressure * nextPressure( )`

**Retourne :**

un pointeur sur un objet `YPressure` accessible en ligne, ou `null` lorsque l'énumération est terminée.

```
pressure->registerTimedReportCallback()pressure  
->registerTimedReportCallback( )
```

YPressure

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YPressureTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**pressure→registerValueCallback()** **pressure→registerValueCallback( )**

**YPressure**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YPressureValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pressure→set\_highestValue()**  
**pressure→setHighestValue()pressure→**  
**set\_highestValue( )**

---

**YPressure**

Modifie la mémoire de valeur maximale observée pour la pression.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la pression

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_logFrequency()**  
**pressure→setLogFrequency()pressure→**  
**set\_logFrequency( )**

**YPressure**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>pressure→set_logicalName()</b>	<b>YPressure</b>
<b>pressure→setLogicalName()pressure→set_logicalName( )</b>	

Modifie le nom logique du capteur de pression.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de pression.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_lowestValue()**

**YPressure**

**pressure→setLowestValue()pressure→  
set\_lowestValue( )**

Modifie la mémoire de valeur minimale observée pour la pression.

**int set\_lowestValue( double newval)**

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la pression

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_reportFrequency()** YPressure  
**pressure→setReportFrequency()pressure→**  
**set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_resolution()****YPressure****pressure→setResolution()pressure→  
set\_resolution( )**

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set(userData)**  
**pressure→setUserData()pressure→**  
**set(userData)**

**YPressure**

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).

**void set(userData void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.30. Interface de la fonction Pwm

La librairie de programmation Yoctopuce permet simplement de configurer, démarrer et arrêter le PWM.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pwmoutput.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YPwmOutput = yoctolib.YPwmOutput;
php	require_once('yocto_pwmoutput.php');
cpp	#include "yocto_pwmoutput.h"
m	#import "yocto_pwmoutput.h"
pas	uses yocto_pwmoutput;
vb	yocto_pwmoutput.vb
cs	yocto_pwmoutput.cs
java	import com.yoctopuce.YoctoAPI.YPwmOutput;
py	from yocto_pwmoutput import *

### Fonction globales

#### yFindPwmOutput(func)

Permet de retrouver un PWM d'après un identifiant donné.

#### yFirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

### Méthodes des objets YPwmOutput

#### pwmoutput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format TYPE ( NAME )=SERIAL.FUNCTIONID.

#### pwmoutput→dutyCycleMove(target, ms\_duration)

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

#### pwmoutput→get\_advertisedValue()

Retourne la valeur courante du PWM (pas plus de 6 caractères).

#### pwmoutput→get\_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

#### pwmoutput→get\_dutyCycleAtPowerOn()

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

#### pwmoutput→get\_enabled()

Retourne l'état de fonctionnement du PWM.

#### pwmoutput→get\_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

#### pwmoutput→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

#### pwmoutput→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

#### pwmoutput→get\_frequency()

Retourne la fréquence du PWM en Hz.

#### pwmoutput→get\_friendlyName()

Retourne un identifiant global du PWM au format NOM\_MODULE . NOM\_FONCTION.

#### pwmoutput→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>pwmoutput→get_functionId()</b> Retourne l'identifiant matériel du PWM, sans référence au module.
<b>pwmoutput→get_hardwareId()</b> Retourne l'identifiant matériel unique du PWM au format SERIAL.FUNCTIONID.
<b>pwmoutput→get_logicalName()</b> Retourne le nom logique du PWM.
<b>pwmoutput→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pwmoutput→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pwmoutput→get_period()</b> Retourne la période du PWM en millisecondes.
<b>pwmoutput→get_pulseDuration()</b> Retourne la longueur d'une impulsion du PWM en millisecondes.
<b>pwmoutput→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>pwmoutput→isOnline()</b> Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.
<b>pwmoutput→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.
<b>pwmoutput→load(msValidity)</b> Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.
<b>pwmoutput→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.
<b>pwmoutput→nextPwmOutput()</b> Continue l'énumération des PWM commencée à l'aide de yFirstPwmOutput().
<b>pwmoutput→pulseDurationMove(ms_target, ms_duration)</b> Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.
<b>pwmoutput→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>pwmoutput→set_dutyCycle(newval)</b> Modifie le duty cycle du PWM, en pour cents.
<b>pwmoutput→set_dutyCycleAtPowerOn(newval)</b> Modifie le duty cycle du PWM au démarrage du module.
<b>pwmoutput→set_enabled(newval)</b> Démarrer ou arrête le PWM.
<b>pwmoutput→set_enabledAtPowerOn(newval)</b> Modifie l'état du fonctionnement du PWM à la mise sous tension du module.
<b>pwmoutput→set_frequency(newval)</b> Modifie la fréquence du PWM.
<b>pwmoutput→set_logicalName(newval)</b> Modifie le nom logique du PWM.
<b>pwmoutput→set_period(newval)</b> Modifie la période du PWM.

**pwmoutput→set\_pulseDuration(newval)**

Modifie la longueur des impulsions du PWM, en millisecondes.

**pwmoutput→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**pwmoutput→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPwmOutput.FindPwmOutput()  
yFindPwmOutput()yFindPwmOutput( )****YPwmOutput**

Permet de retrouver un PWM d'après un identifiant donné.

**YPwmOutput\* yFindPwmOutput( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le PWM soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmOutput.isOnLine()` pour tester si le PWM est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le PWM sans ambiguïté

**Retourne :**

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

**YPwmOutput.FirstPwmOutput()****yFirstPwmOutput()yFirstPwmOutput( )****YPwmOutput**

Commence l'énumération des PWM accessibles par la librairie.

**YPwmOutput\* yFirstPwmOutput( )**

Utiliser la fonction `YPwmOutput.nextPwmOutput( )` pour itérer sur les autres PWM.

**Retourne :**

un pointeur sur un objet `YPwmOutput`, correspondant au premier PWM accessible en ligne, ou `null` si il n'y a pas de PWM disponibles.

**pwmoutput→describe()****YPwmOutput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le PWM (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**pwmoutput→dutyCycleMove()**  
**pwmoutput→dutyCycleMove( )**

**YPwmOutput**

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

int **dutyCycleMove( double target, int ms\_duration)**

**Paramètres :**

**target** nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)

**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→get\_advertisedValue()** YPwmOutput  
**pwmoutput→advertisedValue()** *pwmoutput→get\_advertisedValue( )*

---

Retourne la valeur courante du PWM (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du PWM (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

<b>pwmoutput→get_dutyCycle()</b>	<b>YPwmOutput</b>
<b>pwmoutput→dutyCycle()</b> <b>pwmoutput→get_dutyCycle( )</b>	

---

Retourne le duty cycle du PWM, en pour cents.

```
double get_dutyCycle( )
```

**Retourne :**

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne Y\_DUTYCYCLE\_INVALID.

**pwmoutput→get\_dutyCycleAtPowerOn()** YPwmOutput  
**pwmoutput→dutyCycleAtPowerOn()** pwmoutput→  
**get\_dutyCycleAtPowerOn( )**

---

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

double **get\_dutyCycleAtPowerOn( )**

**Retourne :**

une valeur numérique représentant le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

En cas d'erreur, déclenche une exception ou retourne **Y\_DUTYCYCLETPOWERON\_INVALID**.

**pwmoutput→get\_enabled()**

**YPwmOutput**

**pwmoutput→enabled()pwmoutput→get\_enabled()**

Retourne l'état de fonctionnement du PWM.

**Y\_ENABLED\_enum get\_enabled( )**

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du PWM

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**pwmoutput→get\_enabledAtPowerOn()** YPwmOutput  
**pwmoutput→enabledAtPowerOn()** YPwmOutput→  
**get\_enabledAtPowerOn( )**

---

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

**Y\_ENABLEDATPOWERON\_enum get\_enabledAtPowerOn( )**

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état de fonctionnement du PWM à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

**pwmoutput→get\_errorMessage()**  
**pwmoutput→errorMessage()** **pwmoutput→**  
**get\_errorMessage( )**

**YPwmOutput**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

**pwmoutput→get\_errorType()**  
**pwmoutput→errorType()****pwmoutput→**  
**get\_errorType( )**

**YPwmOutput**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

**pwmoutput→get\_frequency()**  
**pwmoutput→frequency()****pwmoutput→get\_frequency( )**

**YPwmOutput**

Retourne la fréquence du PWM en Hz.

```
int get_frequency( )
```

**Retourne :**

un entier représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne Y\_FREQUENCY\_INVALID.

---

**pwmoutput→get\_friendlyName()** YPwmOutput  
**pwmoutput→friendlyName()** **pwmoutput→get\_friendlyName( )**

---

Retourne un identifiant global du PWM au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du PWM si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du PWM (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le PWM en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**pwmoutput→get\_functionDescriptor()**  
**pwmoutput→functionDescriptor()**  
**pwmoutput→get\_functionDescriptor( )**

**YPwmOutput**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**pwmoutput→get\_functionId()**  
**pwmoutput→functionId()****pwmoutput→**  
**get\_functionId( )**

---

**YPwmOutput**

Retourne l'identifiant matériel du PWM, sans référence au module.

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

<b>pwmoutput→get_hardwareId()</b>	<b>YPwmOutput</b>
<b>pwmoutput→hardwareId()</b> pwmoutput→ <b>get_hardwareId( )</b>	

Retourne l'identifiant matériel unique du PWM au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du PWM (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**pwmoutput→get\_logicalName()**  
**pwmoutput→logicalName()****pwmoutput→get\_logicalName( )**

---

**YPwmOutput**

Retourne le nom logique du PWM.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du PWM. En cas d'erreur, déclenche une exception ou retourne **Y\_LOGICALNAME\_INVALID**.

**pwmoutput→get\_module()**

**YPwmOutput**

**pwmoutput→module()pwmoutput→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule * get_module( )`

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**pwmoutput→get\_period()**

**YPwmOutput**

**pwmoutput→period()pwmoutput→get\_period()**

---

Retourne la période du PWM en millisecondes.

**double get\_period( )**

**Retourne :**

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PERIOD\_INVALID.

**pwmoutput→get\_pulseDuration()**

**YPwmOutput**

**pwmoutput→pulseDuration()pwmoutput→  
get\_pulseDuration()**

Retourne la longueur d'une impulsion du PWM en millisecondes.

**double get\_pulseDuration( )**

**Retourne :**

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PULSEDURATION\_INVALID.

**pwmoutput→get(userData)**  
**pwmoutput→userData()**  
**pwmoutput→get(userData)**

**YPwmOutput**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* get(userData)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pwmoutput→isOnline()****YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du PWM sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le PWM est joignable, false sinon

**pwmoutput→load()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**pwmoutput→nextPwmOutput()**  
pwmoutput→  
**nextPwmOutput( )**

---

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput( )`.

`YPwmOutput * nextPwmOutput( )`

**Retourne :**

un pointeur sur un objet `YPwmOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pwmoutput→pulseDurationMove()**  
**pwmoutput→pulseDurationMove( )****YPwmOutput**

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

```
int pulseDurationMove( double ms_target, int ms_duration)
```

N'importe quel changement de fréquence, duty cycle, période ou encore de longueur d'impulsion annulera tout processus de transition en cours.

**Paramètres :**

**ms\_target** nouvelle longueur des impulsions à la fin de la transition (nombre flottant, représentant la longueur en millisecondes)

**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→registerValueCallback()**  
**pwmoutput→registerValueCallback( )****YPwmOutput**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YPwmOutputValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pwmoutput→set\_dutyCycle()**  
**pwmoutput→setDutyCycle()****pwmoutput→**  
**set\_dutyCycle()**

**YPwmOutput**

Modifie le duty cycle du PWM, en pour cents.

```
int set_dutyCycle( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM, en pour cents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_dutyCycleAtPowerOn()**

**YPwmOutput**

**pwmoutput→setDutyCycleAtPowerOn()** **pwmoutput→**  
**set\_dutyCycleAtPowerOn( )**

Modifie le duty cycle du PWM au démarrage du module.

**int set\_dutyCycleAtPowerOn( double newval)**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM au démarrage du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_enabled()** YPwmOutput  
**pwmoutput→setEnabled()** **pwmoutput→set\_enabled( )**

---

Démarre ou arrête le PWM.

```
int set_enabled( Y_ENABLED_enum newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>pwmoutput→set_enabledAtPowerOn()</b>	<b>YPwmOutput</b>
<b>pwmoutput→setEnabledAtPowerOn()</b> <b>pwmoutput→set_enabledAtPowerOn( )</b>	

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

```
int set_enabledAtPowerOn( Y_ENABLEDATPOWERON_enum newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du fonctionnement du PWM à la mise sous tension du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_frequency()**  
**pwmoutput→setFrequency()****pwmoutput→set\_frequency( )**

**YPwmOutput**

Modifie la fréquence du PWM.

**int set\_frequency( int newval)**

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

**Paramètres :**

**newval** un entier représentant la fréquence du PWM

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_logicalName()****YPwmOutput****pwmoutput→setLogicalName()****pwmoutput→set\_logicalName( )**

Modifie le nom logique du PWM.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du PWM.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_period()**

**YPwmOutput**

**pwmoutput→setPeriod()****pwmoutput→set\_period( )**

---

Modifie la période du PWM.

**int set\_period( double newval)**

**Paramètres :**

**newval** une valeur numérique représentant la période du PWM

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_pulseDuration()**

YPwmOutput

**pwmoutput→setPulseDuration()pwmoutput→  
set\_pulseDuration()**

---

Modifie la longueur des impulsion du PWM, en millisecondes.

```
int set_pulseDuration( double newval)
```

Attention la longueur d'un impulsion ne peut pas être plus grande que la période, dans la cas contraire, la longueur sera automatiquement tronqué à la période.

**Paramètres :**

**newval** une valeur numérique représentant la longueur des impulsion du PWM, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set(userData)**  
**pwmoutput→setUserData()**  
**pwmoutput→set(userData)**

**YPwmOutput**

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).

**void set(userData void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

### 3.31. Interface de la fonction PwmPowerSource

La librairie de programmation Yoctopuce permet de configurer la source de tension utilisée par tous les PWM situés sur un même module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pwmpowersource.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YPwmPowerSource = yoctolib.YPwmPowerSource;
php	require_once('yocto_pwmpowersource.php');
cpp	#include "yocto_pwmpowersource.h"
m	#import "yocto_pwmpowersource.h"
pas	uses yocto_pwmpowersource;
vb	yocto_pwmpowersource.vb
cs	yocto_pwmpowersource.cs
java	import com.yoctopuce.YoctoAPI.YPwmPowerSource;
py	from yocto_pwmpowersource import *

#### Fonction globales

##### yFindPwmPowerSource(func)

Permet de retrouver une source de tension d'après un identifiant donné.

##### yFirstPwmPowerSource()

Commence l'énumération des Source de tension accessibles par la librairie.

#### Méthodes des objets YPwmPowerSource

##### pwmpowersource→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

##### pwmpowersource→get\_advertisedValue()

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

##### pwmpowersource→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

##### pwmpowersource→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

##### pwmpowersource→get\_friendlyName()

Retourne un identifiant global de la source de tension au format NOM\_MODULE . NOM\_FONCTION.

##### pwmpowersource→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

##### pwmpowersource→get\_functionId()

Retourne l'identifiant matériel de la source de tension, sans référence au module.

##### pwmpowersource→get\_hardwareId()

Retourne l'identifiant matériel unique de la source de tension au format SERIAL . FUNCTIONID.

##### pwmpowersource→get\_logicalName()

Retourne le nom logique de la source de tension.

##### pwmpowersource→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

##### pwmpowersource→get\_module\_async(callback, context)

### 3. Reference

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **pwmpowersource→get\_powerMode()**

Retourne la source de tension utilisé par tous les PWM du même module.

#### **pwmpowersource→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **pwmpowersource→isOnline()**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

#### **pwmpowersource→isOnline\_async(callback, context)**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

#### **pwmpowersource→load(msValidity)**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

#### **pwmpowersource→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

#### **pwmpowersource→nextPwmPowerSource()**

Continue l'énumération des Source de tension commencée à l'aide de yFirstPwmPowerSource( ).

#### **pwmpowersource→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **pwmpowersource→set\_logicalName(newval)**

Modifie le nom logique de la source de tension.

#### **pwmpowersource→set\_powerMode(newval)**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

#### **pwmpowersource→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **pwmpowersource→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPwmPowerSource.FindPwmPowerSource()****YPwmPowerSource****yFindPwmPowerSource()yFindPwmPowerSource( )**

Permet de retrouver une source de tension d'après un identifiant donné.

**YPwmPowerSource\* yFindPwmPowerSource( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmPowerSource.isOnLine()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

**Retourne :**

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

**YPwmPowerSource.FirstPwmPowerSource()**  
**yFirstPwmPowerSource()**

---

**YPwmPowerSource**

Commence l'énumération des Source de tension accessibles par la librairie.

**YPwmPowerSource\* yFirstPwmPowerSource( )**

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource( )` pour itérer sur les autres Source de tension.

**Retourne :**

un pointeur sur un objet `YPwmPowerSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de Source de tension disponibles.

**pwmpowersource→describe()**  
**pwmpowersource→  
describe( )****YPwmPowerSource**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant la source de tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pwmpowersource→get\_advertisedValue()**

**YPwmPowerSource**

**pwmpowersource→advertisedValue()**

**pwmpowersource→get\_advertisedValue( )**

---

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**pwmpowersource→get\_errorMessage()**

**YPwmPowerSource**

**pwmpowersource→errorMessage()pwmpowersource  
→get\_errorMessage( )**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

**pwmpowersource→get\_errorType()**  
**pwmpowersource→errorType()**  
**pwmpowersource→get\_errorType( )**

**YPwmPowerSource**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

**YRETCode get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

**pwmpowersource→get\_friendlyName()**

**YPwmPowerSource**

**pwmpowersource→friendlyName()**  
**pwmpowersource  
→get\_friendlyName( )**

Retourne un identifiant global de la source de tension au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et de la source de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la source de tension (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant la source de tension en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**pwmpowersource→get\_functionDescriptor()**  
**pwmpowersource→functionDescriptor()**  
**pwmpowersource→get\_functionDescriptor( )**

---

**YPwmPowerSource**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

`pwmpowersource→get_functionId()`

`YPwmPowerSource`

`pwmpowersource→functionId()pwmpowersource→`  
`get_functionId( )`

---

Retourne l'identifiant matériel de la source de tension, sans référence au module.

`string get_functionId( )`

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la source de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pwmpowersource→get\_hardwareId()**  
**pwmpowersource→hardwareId()** **pwmpowersource→get\_hardwareId( )**

**YPwmPowerSource**

Retourne l'identifiant matériel unique de la source de tension au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la source de tension (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la source de tension (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**pwmpowersource→get\_logicalName()**  
**pwmpowersource→logicalName()pwmpowersource**  
**→get\_logicalName( )**

**YPwmPowerSource**

Retourne le nom logique de la source de tension.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique de la source de tension. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pwmpowersource→get\_module()**

**YPwmPowerSource**

**pwmpowersource→module()** **pwmpowersource→get\_module()**

---

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**pwmpowersource→get\_powerMode()**

**YPwmPowerSource**

**pwmpowersource→powerMode()** **pwmpowersource→get\_powerMode( )**

Retourne la source de tension utilisé par tous les PWM du même module.

**Y\_POWERMODE\_enum get\_powerMode( )**

**Retourne :**

une valeur parmi Y\_POWERMODE\_USB\_5V, Y\_POWERMODE\_USB\_3V, Y\_POWERMODE\_EXT\_V et Y\_POWERMODE\_OPNDRN représentant la source de tension utilisé par tous les PWM du même module

En cas d'erreur, déclenche une exception ou retourne Y\_POWERMODE\_INVALID.

**pwmpowersource→get(userData)**

**YPwmPowerSource**

**pwmpowersource→userData()**  
**pwmpowersource→get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* get(userData)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pwmpowersource→isOnline()**  
**pwmpowersource→  
isOnline( )**

**YPwmPowerSource**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de la source de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si la source de tension est joignable, false sinon

**pwmpowersource→load()****YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**pwmpowersource→nextPwmPowerSource()****YPwmPowerSource****pwmpowersource→nextPwmPowerSource( )**

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

`YPwmPowerSource * nextPwmPowerSource( )`

**Retourne :**

un pointeur sur un objet `YPwmPowerSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pwmpowersource→registerValueCallback()**  
**pwmpowersource→registerValueCallback( )**

**YPwmPowerSource**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YPwmPowerSourceValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pwmpowersource→set\_logicalName()**  
**pwmpowersource→setLogicalName()**  
**pwmpowersource→set\_logicalName( )**

**YPwmPowerSource**

Modifie le nom logique de la source de tension.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la source de tension.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→set\_powerMode()**  
**pwmpowersource→setPowerMode()**  
**pwmpowersource→set\_powerMode( )**

**YPwmPowerSource**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

**int set\_powerMode( Y\_POWERMODE\_enum newval)**

Le PWM peut aussi en mode open drain, dans ce code il tire activement la ligne à zéro volts. Attention ce paramètre est commun à tous les PWM du module, si vous changez le valeur de ce paramètre, tous les PWM situés sur le même module seront affectés. Si vous souhaitez que le changement de ce paramètre soit conservé après un redémarrage du module, n'oubliez pas d'appeler la méthode saveToFlash( ).

**Paramètres :**

**newval** une valeur parmi Y\_POWERMODE\_USB\_5V, Y\_POWERMODE\_USB\_3V, Y\_POWERMODE\_EXT\_V et Y\_POWERMODE\_OPNDRN représentant le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmPowerSource→set(userData)**

**YPwmPowerSource**

**pwmPowerSource→setUserData()** **pwmPowerSource→**  
**set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.32. Interface du quaternion

La class YQt de la librairie Yoctopuce permet d'accéder à l'estimation de l'orientation tridimensionnelle du Yocto-3D sous forme d'un quaternion. Il n'est en général pas nécessaire d'y accéder directement, la classe YGyro offrant une abstraction de plus haut niveau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_gyro.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGyro = yoctolib.YGyro;
require_once('yocto_gyro.php');
cpp #include "yocto_gyro.h"
m #import "yocto_gyro.h"
pas uses yocto_gyro;
vb yocto_gyro.vb
cs yocto_gyro.cs
java import com.yoctopuce.YoctoAPI.YGyro;
py from yocto_gyro import *

```

### Fonction globales

#### **yFindQt(func)**

Permet de retrouver un élément de quaternion d'après un identifiant donné.

#### **yFirstQt()**

Commence l'énumération des éléments de quaternion accessibles par la librairie.

### Méthodes des objets YQt

#### **qt→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **qt→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **qt→get\_advertisedValue()**

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

#### **qt→get\_currentRawValue()**

Retourne la valeur brute renournée par le capteur (sans arrondi ni calibration).

#### **qt→get\_currentValue()**

Retourne la valeur actuelle de la coordonnée.

#### **qt→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

#### **qt→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

#### **qt→get\_friendlyName()**

Retourne un identifiant global de l'élément de quaternion au format NOM\_MODULE . NOM\_FONCTION.

#### **qt→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **qt→get\_functionId()**

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

<b>qt→get_hardwareId()</b>	Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL.FUNCTIONID.
<b>qt→get_highestValue()</b>	Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.
<b>qt→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>qt→get_logicalName()</b>	Retourne le nom logique de l'élément de quaternion.
<b>qt→get_lowestValue()</b>	Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.
<b>qt→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>qt→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>qt→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>qt→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>qt→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>qt→get_unit()</b>	Retourne l'unité dans laquelle la coordonnée est exprimée.
<b>qt→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>qt→isOnline()</b>	Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.
<b>qt→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.
<b>qt→load(msValidity)</b>	Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.
<b>qt→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>qt→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.
<b>qt→nextQt()</b>	Continue l'énumération des éléments de quaternion commencée à l'aide de yFirstQt( ).
<b>qt→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>qt→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>qt→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.

**qt→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**qt→set\_logicalName(newval)**

Modifie le nom logique de l'élément de quaternion.

**qt→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**qt→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**qt→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**qt→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**qt→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YQt.FindQt()****YQt****yFindQt()yFindQt( )**

Permet de retrouver un élément de quaternion d'après un identifiant donné.

**YQt\* yFindQt( string func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnline()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

**Retourne :**

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

## **YQt.FirstQt() yFirstQt()yFirstQt( )**

---

**YQt**

Commence l'énumération des éléments de quaternion accessibles par la librairie.

**YQt\* yFirstQt( )**

Utiliser la fonction `YQt .nextQt( )` pour itérer sur les autres éléments de quaternion.

**Retourne :**

un pointeur sur un objet `YQt`, correspondant au premier élément de quaternion accessible en ligne, ou `null` si il n'y a pas de éléments de quaternion disponibles.

**qt→calibrateFromPoints()qt→  
calibrateFromPoints( )****YQt**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→describe()qt→describe( )****YQt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format TYPE (NAME )=SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant l'élément de quaternion (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**qt→get\_advertisedValue()****YQt****qt→advertisedValue()qt→get\_advertisedValue()**

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

```
string get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

<b>qt→get_currentRawValue()</b>	<b>YQt</b>
<b>qt→currentRawValue()qt→ get_currentRawValue( )</b>	

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
double get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

---

**qt→get\_currentValue()**  
**qt→currentValue()qt→get\_currentValue()****YQt**

Retourne la valeur actuelle de la coordonnée.

```
double get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la coordonnée

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

---

<b>qt→get_errorMessage()</b>	<b>YQt</b>
<b>qt→errorMessage()qt→get_errorMessage( )</b>	

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

---

**qt→get\_errorType()**  
**qt→errorType()qt→get\_errorType( )****YQt**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

---

<b>qt→get_friendlyName()</b>	<b>YQt</b>
<b>qt→friendlyName()qt→get_friendlyName( )</b>	

---

Retourne un identifiant global de l'élément de quaternion au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et de l'élément de quaternion si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'élément de quaternion (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**qt→get\_functionDescriptor()**  
**qt→functionDescriptor()qt→**  
**get\_functionDescriptor( )**

YQt

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

`qt->get_functionId()`

YQt

`qt->functionId()qt->get_functionId()`

---

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

`string get_functionId( )`

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**qt→get\_hardwareId()****YQt****qt→hardwareId()qt→get\_hardwareId( )**

Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'élément de quaternion (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**qt→get\_highestValue()**

**YQt**

**qt→highestValue()qt→get\_highestValue( )**

---

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

double **get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**qt→get\_logFrequency()****YQt****qt→logFrequency()qt→get\_logFrequency( )**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
string get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**qt→get\_logicalName()**

YQt

**qt→logicalName()qt→get\_logicalName( )**

---

Retourne le nom logique de l'élément de quaternion.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique de l'élément de quaternion. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**qt→get\_lowestValue()****YQt****qt→lowestValue()qt→get\_lowestValue( )**

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

**double get\_lowestValue( )****Retourne :**

une valeur numérique représentant la valeur minimale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**qt→get\_module()**

**YQt**

**qt→module()qt→get\_module( )**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**qt→get\_recordedData()****YQt****qt→recordedData()qt→get\_recordedData( )**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

<b>qt→get_reportFrequency()</b>	<b>YQt</b>
<b>qt→reportFrequency()qt→get_reportFrequency( )</b>	

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
string get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

**qt→get\_resolution()**  
**qt→resolution()qt→get\_resolution( )****YQt**

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**qt→get\_unit()**

**YQt**

**qt→unit()qt→get\_unit()**

---

Retourne l'unité dans laquelle la coordonnée est exprimée.

string **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la coordonnée est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**qt→get(userData)****YQt****qt→userData()qt→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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.

**qt→isOnline()**

YQt

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'élément de quaternion est joignable, false sinon

**qt→load()qt→load( )****YQt**

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→loadCalibrationPoints()qt→  
loadCalibrationPoints()****YQt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt->nextQt()qt->nextQt( )****YQt**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

`YQt * nextQt( )`

**Retourne :**

un pointeur sur un objet `YQt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**qt→registerTimedReportCallback()qt→  
registerTimedReportCallback( )****YQt**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YQtTimedReportCallback callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**qt→registerValueCallback()qt→  
registerValueCallback( )****YQt**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YQtValueCallback 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 appellés 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.

---

**YQt**  
**qt→set\_highestValue()**  
**qt→setHighestValue()qt→set\_highestValue( )**

---

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logFrequency()**

YQt

**qt→setLogFrequency()qt→set\_logFrequency( )**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( const string& newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>qt→set_logicalName()</b>	<b>YQt</b>
<b>qt→setLogicalName()qt→set_logicalName( )</b>	

---

Modifie le nom logique de l'élément de quaternion.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'élément de quaternion.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**qt→set\_lowestValue()**  
**qt→setLowestValue()qt→set\_lowestValue( )**

YQt

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double 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.

<b>qt→set_reportFrequency()</b>	<b>YQt</b>
<b>qt→setReportFrequency()qt→</b>	

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_resolution()****YQt****qt→setResolution()qt→set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set(userData)****YQt****qt→setUserData()qt→set(userData)()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData void* 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

### 3.33. Interface de la fonction Horloge Temps Réel

La fonction RealTimeClock fournit la date et l'heure courante de manière persistante, même en cas de coupure de courant de plusieurs jours. Elle est le fondement des fonctions de réveil automatique implémentées par le WakeUpScheduler. L'heure courante peut représenter aussi bien une heure locale qu'une heure UTC, mais aucune adaptation automatique n'est faite au changement d'heure été/hiver.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_realtimedclock.js'></script>
node.js var yoctolib = require('yoctolib');
var YRealTimeClock = yoctolib.YRealTimeClock;
php require_once('yocto_realtimedclock.php');
cpp #include "yocto_realtimedclock.h"
m #import "yocto_realtimedclock.h"
pas uses yocto_realtimedclock;
vb yocto_realtimedclock.vb
cs yocto_realtimedclock.cs
java import com.yoctopuce.YoctoAPI.YRealTimeClock;
py from yocto_realtimedclock import *

```

#### Fonction globales

##### **yFindRealTimeClock(func)**

Permet de retrouver une horloge d'après un identifiant donné.

##### **yFirstRealTimeClock()**

Commence l'énumération des horloges accessibles par la librairie.

#### Méthodes des objets YRealTimeClock

##### **realtimeclock→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format TYPE (NAME) = SERIAL.FUNCTIONID.

##### **realtimeclock→get\_advertisedValue()**

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

##### **realtimeclock→get\_dateTime()**

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

##### **realtimeclock→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

##### **realtimeclock→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

##### **realtimeclock→get\_friendlyName()**

Retourne un identifiant global de l'horloge au format NOM\_MODULE.NOM\_FONCTION.

##### **realtimeclock→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

##### **realtimeclock→get\_functionId()**

Retourne l'identifiant matériel de l'horloge, sans référence au module.

##### **realtimeclock→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'horloge au format SERIAL.FUNCTIONID.

##### **realtimeclock→get\_logicalName()**

Retourne le nom logique de l'horloge.

##### **realtimeclock→get\_module()**

### 3. Reference

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **realtimeclock→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **realtimeclock→get\_timeSet()**

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

#### **realtimeclock→get\_unixTime()**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

#### **realtimeclock→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **realtimeclock→get\_utcOffset()**

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

#### **realtimeclock→isOnline()**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

#### **realtimeclock→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

#### **realtimeclock→load(msValidity)**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

#### **realtimeclock→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

#### **realtimeclock→nextRealTimeClock()**

Continue l'énumération des horloge commencée à l'aide de yFirstRealTimeClock( ).

#### **realtimeclock→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **realtimeclock→set\_logicalName(newval)**

Modifie le nom logique de l'horloge.

#### **realtimeclock→set\_unixTime(newval)**

Modifie l'heure courante.

#### **realtimeclock→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **realtimeclock→set\_utcOffset(newval)**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

#### **realtimeclock→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YRealTimeClock.FindRealTimeClock()****YRealTimeClock****yFindRealTimeClock()yFindRealTimeClock( )**

Permet de retrouver une horloge d'après un identifiant donné.

**YRealTimeClock\* yFindRealTimeClock( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnline()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'horloge sans ambiguïté

**Retourne :**

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

## **YRealTimeClock.FirstRealTimeClock()**

## **YRealTimeClock**

### **yFirstRealTimeClock()yFirstRealTimeClock( )**

Commence l'énumération des horloge accessibles par la librairie.

**YRealTimeClock\* yFirstRealTimeClock( )**

Utiliser la fonction `YRealTimeClock.nextRealTimeClock()` pour itérer sur les autres horloge.

**Retourne :**

un pointeur sur un objet `YRealTimeClock`, correspondant à la première horloge accessible en ligne, ou `null` si il n'y a pas de horloge disponibles.

**realtimeclock→describe()** **realtimeclock→  
describe( )****YRealTimeClock**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format TYPE ( NAME ) = SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'horloge (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**realtimeclock→get\_advertisedValue()** **YRealTimeClock**  
**realtimeclock→advertisedValue()realtimeclock→**  
**get\_advertisedValue( )**

---

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

---

<b>realtimeclock→getDateTime()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→dateTime()realtimeclock→getDateTime( )</b>	

---

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

```
string getDateTime( )
```

**Retourne :**

une chaîne de caractères représentant l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

En cas d'erreur, déclenche une exception ou retourne Y\_DATETIME\_INVALID.

**realtimeclock→getErrorMessage()**

**YRealTimeClock**

**realtimeclock→errorMessage()realtimeclock→  
getErrorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

---

**realtimeclock→get\_errorType()**  
**realtimeclock→errorType()****realtimeclock→get\_errorType( )**

**YRealTimeClock**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

**realtimeclock→get\_friendlyName()** **YRealTimeClock**  
**realtimeclock→friendlyName()realtimeclock→**  
**get\_friendlyName( )**

Retourne un identifiant global de l'horloge au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et de l'horloge si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'horloge (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'horloge en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

---

<b>realtimeclock→get_functionDescriptor()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→functionDescriptor()realtimeclock →get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**realtimeclock→get\_functionId()** **YRealTimeClock**  
**realtimeclock→functionId()realtimeclock→**  
**get\_functionId( )**

---

Retourne l'identifiant matériel de l'horloge, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

<b>realtimeclock→get_hardwareId()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→hardwareId()realtimeclock→get_hardwareId( )</b>	

---

Retourne l'identifiant matériel unique de l'horloge au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'horloge (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

<b>realtimeclock→get_logicalName()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→logicalName()</b>	
<b>realtimeclock→get_logicalName( )</b>	

---

Retourne le nom logique de l'horloge.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'horloge. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**realtimeclock→get\_module()**  
**realtimeclock→module()realtimeclock→**  
**get\_module()**

**YRealTimeClock**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** rentrée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**realtimeclock→get\_timeSet()**  
**realtimeclock→timeSet()realtimeclock→**  
**get\_timeSet( )**

---

**YRealTimeClock**

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

**Y\_TIMESET\_enum get\_timeSet( )**

**Retourne :**

soit Y\_TIMESET\_FALSE, soit Y\_TIMESET\_TRUE, selon vrai si l'horloge à été mise à l'heure, sinon faux

En cas d'erreur, déclenche une exception ou retourne Y\_TIMESET\_INVALID.

**realtimeclock→get\_unixTime()**

**YRealTimeClock**

**realtimeclock→unixTime()realtimeclock→  
get\_unixTime( )**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

**s64 get\_unixTime( )**

**Retourne :**

un entier représentant l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970)

En cas d'erreur, déclenche une exception ou retourne Y\_UNIXTIME\_INVALID.

**realtimeclock→get(userData())** **YRealTimeClock**  
**realtimeclock→userData()realtimeclock→**  
**get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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.

---

<b>realtimeclock→get_utcOffset()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→utcOffset()realtimeclock→get_utcOffset( )</b>	

---

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

```
int get_utcOffset( )
```

**Retourne :**

un entier représentant le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne Y\_UTCOFFSET\_INVALID.

**realtimeclock→isOnline()** **realtimeclock→**  
**isOnline( )**

---

**YRealTimeClock**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'horloge est joignable, false sinon

**realtimeclock→load()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock**→**nextRealTimeClock()**  
**realtimeclock**  
→**nextRealTimeClock( )**

**YRealTimeClock**

Continue l'énumération des horloge commencée à l'aide de **yFirstRealTimeClock( )**.

**YRealTimeClock \* nextRealTimeClock( )**

**Retourne :**

un pointeur sur un objet **YRealTimeClock** accessible en ligne, ou **null** lorsque l'énumération est terminée.

---

**realtimeclock→registerValueCallback()**  
**realtimeclock→registerValueCallback( )****YRealTimeClock**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YRealTimeClockValueCallback 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 appellés 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.

<b>realtimeclock→set_logicalName()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→setLogicalName()realtimeclock→set_logicalName( )</b>	

Modifie le nom logique de l'horloge.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'horloge.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>realtimeclock→set_unixTime()</b>	<b>YRealTimeClock</b>
<b>realtimeclock→setUnixTime()</b> <b>realtimeclock→set_unixTime( )</b>	

Modifie l'heure courante.

**int set\_unixTime( s64 newval)**

L'heure est passée au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970). Si l'heure UTC est connue, l'attribut utcOffset sera automatiquement ajusté en fonction de l'heure configurée.

**Paramètres :**

**newval** un entier représentant l'heure courante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→set(userData())** **YRealTimeClock**  
**realtimeclock→setUserData()realtimeclock→**  
**set(userData())**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**realtimeclock→set\_utcOffset()****YRealTimeClock****realtimeclock→setUtcOffset()realtimeclock→  
set\_utcOffset( )**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
int set_utcOffset( int newval)
```

Le décallage est automatiquement arrondi au quart d'heure le plus proche. Si l'heure UTC est connue, l'heure courante sera automatiquement adaptée en fonction du décalage choisi.

**Paramètres :**

**newval** un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.34. Configuration du référentiel

Cette classe permet de configurer l'orientation dans laquelle le Yocto-3D est utilisé, afin que les fonctions d'orientation relatives au plan de la surface terrestre utilisent le référentiel approprié. La classe offre aussi un processus de recalibration tridimensionnel des capteurs, permettant de compenser les variations locales de l'accélération terrestre et d'améliorer la précision des capteurs d'inclinaisons.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_refframe.js'></script>
nodejs var yoctolib = require('yoctolib');
var YRefFrame = yoctolib.YRefFrame;
php require_once('yocto_refframe.php');
cpp #include "yocto_refframe.h"
m #import "yocto_refframe.h"
pas uses yocto_refframe;
vb yocto_refframe.vb
cs yocto_refframe.cs
java import com.yoctopuce.YoctoAPI.YRefFrame;
py from yocto_refframe import *

```

### Fonction globales

#### yFindRefFrame(func)

Permet de retrouver un référentiel d'après un identifiant donné.

#### yFirstRefFrame()

Commence l'énumération des référentiels accessibles par la librairie.

### Méthodes des objets YRefFrame

#### refframe→cancel3DCalibration()

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

#### refframe→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### refframe→get\_3DCalibrationHint()

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode start3DCalibration.

#### refframe→get\_3DCalibrationLogMsg()

Retourne le dernier message de log produit par le processus de calibration.

#### refframe→get\_3DCalibrationProgress()

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode start3DCalibration.

#### refframe→get\_3DCalibrationStage()

Retourne l'index de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

#### refframe→get\_3DCalibrationStageProgress()

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

#### refframe→get\_advertisedValue()

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

#### refframe→get\_bearing()

Retourne le cap de référence utilisé par le compas.

#### **refframe→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

#### **refframe→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

#### **refframe→get\_friendlyName()**

Retourne un identifiant global du référentiel au format NOM\_MODULE . NOM\_FONCTION.

#### **refframe→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **refframe→get\_functionId()**

Retourne l'identifiant matériel du référentiel, sans référence au module.

#### **refframe→get\_hardwareId()**

Retourne l'identifiant matériel unique du référentiel au format SERIAL . FUNCTIONID.

#### **refframe→get\_logicalName()**

Retourne le nom logique du référentiel.

#### **refframe→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **refframe→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **refframe→get\_mountOrientation()**

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

#### **refframe→get\_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

#### **refframe→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **refframe→isOnline()**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

#### **refframe→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

#### **refframe→load(msValidity)**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

#### **refframe→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

#### **refframe→more3DCalibration()**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.

#### **refframe→nextRefFrame()**

Continue l'énumération des référentiels commencée à l'aide de yFirstRefFrame( ).

#### **refframe→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **refframe→save3DCalibration()**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

#### **refframe→set\_bearing(newval)**

### 3. Reference

---

Modifie le cap de référence utilisé par le compas.

**refframe→set\_logicalName(newval)**

Modifie le nom logique du référentiel.

**refframe→set\_mountPosition(position, orientation)**

Modifie le référentiel de la boussole et des inclinomètres.

**refframe→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**refframe→start3DCalibration()**

Initie le processus de calibration tridimensionnelle des capteurs.

**refframe→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YRefFrame.FindRefFrame()****YRefFrame****yFindRefFrame()yFindRefFrame( )**

Permet de retrouver un référentiel d'après un identifiant donné.

**YRefFrame\* yFindRefFrame( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRefFrame.isOnline()` pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le référentiel sans ambiguïté

**Retourne :**

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

### **YRefFrame.FirstRefFrame()**

**YRefFrame**

### **yFirstRefFrame()yFirstRefFrame( )**

Commence l'énumération des référentiels accessibles par la librairie.

**YRefFrame\* yFirstRefFrame( )**

Utiliser la fonction `YRefFrame.nextRefFrame( )` pour itérer sur les autres référentiels.

**Retourne :**

un pointeur sur un objet `YRefFrame`, correspondant au premier référentiel accessible en ligne, ou `null` si il n'y a pas de référentiels disponibles.

---

**refframe**→**cancel3DCalibration()**  
**refframe**→  
**cancel3DCalibration( )****YRefFrame**

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

int **cancel3DCalibration( )**

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→describe()****YRefFrame**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format TYPE ( NAME )=SERIAL.FUNCTIONID.

string **describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le référentiel (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**refframe→get\_3DCalibrationHint()**  
**refframe→3DCalibrationHint()refframe→**  
**get\_3DCalibrationHint()**

**YRefFrame**

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode start3DCalibration.

**string get\_3DCalibrationHint( )**

**Retourne :**

une chaîne de caractères.

**refframe→get\_3DCalibrationLogMsg()** YRefFrame  
**refframe→3DCalibrationLogMsg()** **refframe→**  
**get\_3DCalibrationLogMsg( )**

---

Retourne le dernier message de log produit par le processus de calibration.

**string get\_3DCalibrationLogMsg( )**

Si aucun nouveau message n'est disponible, retourne une chaîne vide.

**Retourne :**  
une chaîne de caractères.

---

```
refframe→get_3DCalibrationProgress()  
refframe→3DCalibrationProgress()refframe→  
get_3DCalibrationProgress( )
```

**YRefFrame**

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode start3DCalibration.

```
int get_3DCalibrationProgress( )
```

**Retourne :**

une nombre entier entre 0 (pas commencé) et 100 (terminé).

**refframe→get\_3DCalibrationStage()**  
**refframe→3DCalibrationStage()refframe→**  
**get\_3DCalibrationStage( )**

**YRefFrame**

Retourne l'index de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

**int get\_3DCalibrationStage( )**

**Retourne :**

une nombre entier, croissant au fur et à mesure de la complétion des étapes.

**refframe→get\_3DCalibrationStageProgress()**

**YRefFrame**

**refframe→3DCalibrationStageProgress()refframe→  
get\_3DCalibrationStageProgress( )**

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

**int get\_3DCalibrationStageProgress( )**

**Retourne :**

une nombre entier entre 0 (pas commencé) et 100 (terminé).

**refframe→get\_advertisedValue()**  
**refframe→advertisedValue()refframe→**  
**get\_advertisedValue( )**

---

**YRefFrame**

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du référentiel (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**refframe→get\_bearing()****YRefFrame****refframe→bearing()refframe→get\_bearing( )**

Retourne le cap de référence utilisé par le compas.

```
double get_bearing( )
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici.

**Retourne :**

une valeur numérique représentant le cap de référence utilisé par le compas

En cas d'erreur, déclenche une exception ou retourne Y\_BEARING\_INVALID.

**refframe→get\_errorMessage()**  
**refframe→errorMessage()refframe→**  
**get\_errorMessage( )**

---

**YRefFrame**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

---

**refframe→get\_errorType()****YRefFrame****refframe→errorType()refframe→get\_errorType( )**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

**refframe→get\_friendlyName()**  
**refframe→friendlyName()** **refframe→get\_friendlyName( )**

---

**YRefFrame**

Retourne un identifiant global du référentiel au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du référentiel si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du référentiel (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le référentiel en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**refframe→get\_functionDescriptor()**  
**refframe→functionDescriptor()** **refframe→get\_functionDescriptor( )**

**YRefFrame**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>refframe→get_functionId()</b>	<b>YRefFrame</b>
<b>refframe→functionId()</b>	<b>refframe→</b>
<b>get_functionId( )</b>	

---

Retourne l'identifiant matériel du référentiel, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

**refframe**→**get\_hardwareId()**  
**refframe**→**hardwareId()****refframe**→  
**get\_hardwareId( )**

**YRefFrame**

Retourne l'identifiant matériel unique du référentiel au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du référentiel (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**refframe→get\_logicalName()**  
**refframe→logicalName()**refframe→  
**get\_logicalName( )**

---

**YRefFrame**

Retourne le nom logique du référentiel.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du référentiel. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**refframe→get\_module()****YRefFrame****refframe→module()refframe→get\_module( )**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

---

<b>refframe→get_mountOrientation()</b>	<b>YRefFrame</b>
<b>refframe→mountOrientation()</b> <b>refframe→get_mountOrientation( )</b>	

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

**Y\_MOUNTORIENTATION get\_mountOrientation( )**

**Retourne :**

une valeur parmi l'énumération Y\_MOUNTORIENTATION (Y\_MOUNTORIENTATION\_TWELVE, Y\_MOUNTORIENTATION\_THREE, Y\_MOUNTORIENTATION\_SIX, Y\_MOUNTORIENTATION\_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get\_mountPosition()**  
**refframe→mountPosition()refframe→**  
**get\_mountPosition()**

**YRefFrame**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

**Y\_MOUNTPOSITION get\_mountPosition( )**

**Retourne :**

une valeur parmi l'énumération Y\_MOUNTPOSITION (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), correspondant à l'installation dans une boîte, sur l'une des six faces

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get(userData)**

**YRefFrame**

**refframe→userData()refframe→get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* get(userData) ( )**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**refframe→isOnline()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du référentiel sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le référentiel est joignable, false sinon

**refframe→load()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe**→**more3DCalibration()****refframe**→  
**more3DCalibration()**

**YRefFrame**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

```
int more3DCalibration( )
```

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode `get_3DCalibrationHint` (les instructions changent pendant la procédure de calibration). En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe**→**nextRefFrame()****refframe**→  
**nextRefFrame( )**

---

**YRefFrame**

Continue l'énumération des référentiels commencée à l'aide de **yFirstRefFrame( )**.

**YRefFrame \* nextRefFrame( )**

**Retourne :**

un pointeur sur un objet **YRefFrame** accessible en ligne, ou **null** lorsque l'énumération est terminée.

**refframe→registerValueCallback()****YRefFrame**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YRefFrameValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**refframe→save3DCalibration()**  
**refframe→  
save3DCalibration( )**

---

**YRefFrame**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

**int save3DCalibration( )**

N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé après le redémarrage du module. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_bearing()****YRefFrame****refframe→setBearing()refframe→set\_bearing()**

Modifie le cap de référence utilisé par le compas.

```
int set_bearing( double newval)
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici. Par exemple, si vous indiquez comme cap de référence la valeur de la déclinaison magnétique terrestre, le compas donnera l'orientation par rapport au Nord géographique. De même, si le capteur n'est pas positionné dans une des directions standard à cause d'un angle de lacet supplémentaire, vous pouvez le configurer comme cap de référence afin que le compas donne la direction naturelle attendue.

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur numérique représentant le cap de référence utilisé par le compas

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>refframe→set_logicalName()</b>	<b>YRefFrame</b>
<b>refframe→setLogicalName()</b> <b>refframe→set_logicalName( )</b>	

---

Modifie le nom logique du référentiel.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du référentiel.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_mountPosition()**  
**refframe→setMountPosition()** **refframe→set\_mountPosition()**

**YRefFrame**

Modifie le référentiel de la boussole et des inclinomètres.

```
int set_mountPosition( Y_MOUNTPOSITION position,
                      Y_MOUNTORIENTATION orientation)
```

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas où le module n'est pas utilisé horizontalement et à l'endroit, il faut indiquer son orientation de référence (parallèle à la surface terrestre) afin que les mesures soient faites relativement à cette position.

**Paramètres :**

**position** une valeur parmi l'énumération Y\_MOUNTPOSITION (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), correspondant à l'installation dans une boîte, sur l'une des six faces.

**orientation** une valeur parmi l'énumération Y\_MOUNTORIENTATION (Y\_MOUNTORIENTATION\_TWELVE, Y\_MOUNTORIENTATION\_THREE, Y\_MOUNTORIENTATION\_SIX, Y\_MOUNTORIENTATION\_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière. N'oubliez pas d'appeler la méthode saveToFlash() du module si le réglage doit être préservé.

**refframe→set(userData())  
refframe→setUserData()** **refframe→  
set(userData())**

**YRefFrame**

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).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**refframe**→**start3DCalibration()****refframe**→  
**start3DCalibration( )**

---

**YRefFrame**

Initie le processus de calibration tridimensionnelle des capteurs.

**int start3DCalibration( )**

Cette calibration est utilisée à bas niveau pour l'estimation innertielle de position et pour améliorer la précision des mesures d'inclinaison. Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.35. Interface de la fonction Relay

La librairie de programmation Yoctopuce permet simplement de changer l'état du relais. Le changement d'état n'est pas persistant: le relais retournera spontanément à sa position de repos dès que le module est mis hors tension ou redémarré. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Pour les modules dotés de deux sorties par relais (relai inverseur), les deux sorties sont appelées A et B, la sortie A correspondant à la position de repos (hors tension) et la sortie B correspondant à l'état actif. Si vous préféreriez l'état par défaut opposé, vous pouvez simplement changer vos fils sur le bornier.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_relay.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YRelay = yoctolib.YRelay;
php	require_once('yocto_relay.php');
cpp	#include "yocto_relay.h"
m	#import "yocto_relay.h"
pas	uses yocto_relay;
vb	yocto_relay.vb
cs	yocto_relay.cs
java	import com.yoctopuce.YoctoAPI.YRelay;
py	from yocto_relay import *

### Fonction globales

#### yFindRelay(func)

Permet de retrouver un relais d'après un identifiant donné.

#### yFirstRelay()

Commence l'énumération des relais accessibles par la librairie.

### Méthodes des objets YRelay

#### relay->delayedPulse(ms\_delay, ms\_duration)

Pré-programme une impulsion

#### relay->describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format TYPE(NAME)=SERIAL.FUNCTIONID.

#### relay->get\_advertisedValue()

Retourne la valeur courante du relais (pas plus de 6 caractères).

#### relay->get\_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

#### relay->get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

#### relay->get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

#### relay->get\_friendlyName()

Retourne un identifiant global du relais au format NOM\_MODULE.NOM\_FONCTION.

#### relay->get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### relay->get\_functionId()

Retourne l'identifiant matériel du relais, sans référence au module.

<b>relay→get_hardwareId()</b>	Retourne l'identifiant matériel unique du relais au format SERIAL.FUNCTIONID.
<b>relay→get_logicalName()</b>	Retourne le nom logique du relais.
<b>relay→get_maxTimeOnStateA()</b>	Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.
<b>relay→get_maxTimeOnStateB()</b>	Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.
<b>relay→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>relay→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>relay→get_output()</b>	Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.
<b>relay→get_pulseTimer()</b>	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_stateAtPowerOn()</b>	Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).
<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 le relais est joignable, sans déclencher d'erreur.
<b>relay→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.
<b>relay→load(msValidity)</b>	Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.
<b>relay→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du relais, 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_maxTimeOnStateA(newval)</b>	Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.
<b>relay→set_maxTimeOnStateB(newval)</b>	

### 3. Reference

---

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

#### **relay→set\_output(newval)**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

#### **relay→set\_state(newval)**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

#### **relay→set\_stateAtPowerOn(newval)**

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

#### **relay→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **relay→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YRelay.FindRelay()

## YRelay

### yFindRelay()yFindRelay( )

Permet de retrouver un relais d'après un identifiant donné.

YRelay\* **yFindRelay( const string& 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()

YRelay

### yFirstRelay()yFirstRelay()

---

Commence l'énumération des relais accessibles par la librairie.

YRelay\* **yFirstRelay( )**

Utiliser la fonction YRelay.nextRelay( ) pour itérer sur les autres relais.

**Retourne :**

un pointeur sur un objet YRelay, correspondant au premier relais accessible en ligne, ou null si il n'y a pas de relais disponibles.

**relay->delayedPulse()****YRelay**

Pré-programme une impulsion

```
int delayedPulse( int ms_delay, int ms_duration)
```

**Paramètres :**

**ms\_delay** délai d'attente avant l'impulsion, en millisecondes

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay->describe()****YRelay**

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format TYPE ( NAME )=SERIAL.FUNCTIONID.

string **describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le relais (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**relay→get\_advertisedValue()**  
**relay→advertisedValue()relay→**  
**get\_advertisedValue( )**

**YRelay**

Retourne la valeur courante du relais (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du relais (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**relay→get\_countdown()****YRelay****relay→countdown()relay→get\_countdown( )**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

**s64 get\_countdown( )**

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne Y\_COUNTDOWN\_INVALID.

**relay→get\_errorMessage()****YRelay****relay→errorMessage()relay→get\_errorMessage( )**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du relais.

**relay->get\_errorType()** YRelay  
**relay->errorType()relay->get\_errorType( )**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du relais.

**relay→get\_friendlyName()****YRelay****relay→friendlyName()relay→get\_friendlyName( )**

Retourne un identifiant global du relais au format NOM\_MODULE.NOM\_FONCTION.

```
string get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du relais si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du relais (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le relais en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

<b>relay-&gt;get_functionDescriptor()</b>	<b>YRelay</b>
<b>relay-&gt;functionDescriptor()relay-&gt;</b>	
<b>get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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

---

<b>relay-&gt;get_functionId()</b>	<b>YRelay</b>
<b>relay-&gt;functionId()relay-&gt;get_functionId()</b>	

---

Retourne l'identifiant matériel du relais, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le relais (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

<b>relay→get_hardwareId()</b>	<b>YRelay</b>
<b>relay→hardwareId()relay→get_hardwareId()</b>	

---

Retourne l'identifiant matériel unique du relais au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du relais (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le relais (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**relay→get\_logicalName()****YRelay****relay→logicalName()relay→get\_logicalName( )**

Retourne le nom logique du relais.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du relais. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**relay→get\_maxTimeOnStateA()**  
**relay→maxTimeOnStateA()relay→**  
**get\_maxTimeOnStateA( )**

**YRelay**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**s64 get\_maxTimeOnStateA( )**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEA\_INVALID.

**relay→get\_maxTimeOnStateB()**  
**relay→maxTimeOnStateB()relay→**  
**get\_maxTimeOnStateB( )**

**YRelay**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

s64 **get\_maxTimeOnStateB( )**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEB\_INVALID.

**relay→get\_module()** YRelay  
**relay→module()relay→get\_module()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**relay→get\_output()****YRelay****relay→output()relay→get\_output( )**

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

**Y\_OUTPUT\_enum 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.

---

<b>relay→get_pulseTimer()</b>	<b>YRelay</b>
<b>relay→pulseTimer()relay→get_pulseTimer( )</b>	

---

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

s64 **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\_PULSE\_TIMER\_INVALID.

**relay→get\_state()****YRelay****relay→state()relay→get\_state( )**

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

**Y\_STATE\_enum 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.

<b>relay→get_stateAtPowerOn()</b>	<b>YRelay</b>
<b>relay→stateAtPowerOn()relay→get_stateAtPowerOn( )</b>	

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

**Y\_STATEATPOWERON\_enum get\_stateAtPowerOn( )**

**Retourne :**

une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B représentant l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y\_STATEATPOWERON\_INVALID.

**relay→get(userData)****YRelay****relay→userData()relay→get(userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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()**

YRelay

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le relais est joignable, false sinon

**relay->load()****YRelay**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **relay→nextRelay()**

**YRelay**

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

`YRelay * nextRelay( )`

**Retourne :**

un pointeur sur un objet `YRelay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**relay->pulse()  
relay->pulse( )****YRelay**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
int pulse( int ms_duration)
```

**Paramètres :**

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

```
relay->registerValueCallback()relay->  
registerValueCallback( )
```

YRelay

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YRelayValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**relay→set\_logicalName()**  
**relay→setLogicalName()**relay→  
**set\_logicalName( )**

**YRelay**

Modifie le nom logique du relais.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du relais.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

```
relay→set_maxTimeOnStateA()  
relay→setMaxTimeOnStateA()relay→  
set_maxTimeOnStateA( )
```

YRelay

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
int set_maxTimeOnStateA( s64 newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

```
relay->set_maxTimeOnStateB()  
relay->setMaxTimeOnStateB()relay->  
set_maxTimeOnStateB( )
```

YRelay

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
int set_maxTimeOnStateB( s64 newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_output()**

**YRelay**

**relay→setOutput()relay→set\_output( )**

---

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
int set_output( Y_OUTPUT_enum newval)
```

**Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**relay->set\_state()** YRelay  
**relay->setState()relay->set\_state( )**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
int set_state( Y_STATE_enum 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.

<b>relay→set_stateAtPowerOn()</b>	<b>YRelay</b>
<b>relay→setStateAtPowerOn()relay→</b>	
<b>set_stateAtPowerOn( )</b>	

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
int set_stateAtPowerOn( Y_STATEATPOWERON_enum newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set(userData)****YRelay****relay→setUserData()relay→set(userData())**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.36. Interface des fonctions de type senseur

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Fonction globales

#### **yFindSensor(func)**

Permet de retrouver un senseur d'après un identifiant donné.

#### **yFirstSensor()**

Commence l'énumération des senseurs accessibles par la librairie.

### Méthodes des objets YSensor

#### **sensor->calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **sensor->describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format TYPE(NAME)=SERIAL.FUNCTIONID.

#### **sensor->get\_advertisedValue()**

Retourne la valeur courante du senseur (pas plus de 6 caractères).

#### **sensor->get\_currentRawValue()**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

#### **sensor->get\_currentValue()**

Retourne la valeur actuelle de la mesure.

#### **sensor->get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

#### **sensor->get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

#### **sensor->get\_friendlyName()**

Retourne un identifiant global du senseur au format NOM\_MODULE.NOM\_FONCTION.

#### **sensor->get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **sensor->get\_functionId()**

Retourne l'identifiant matériel du senseur, sans référence au module.

#### **sensor->get\_hardwareId()**

Retourne l'identifiant matériel unique du senseur au format SERIAL . FUNCTIONID.
<b>sensor→get_highestValue()</b>
Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.
<b>sensor→get_logFrequency()</b>
Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>sensor→get_logicalName()</b>
Retourne le nom logique du senseur.
<b>sensor→get_lowestValue()</b>
Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.
<b>sensor→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>sensor→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>sensor→get_recordedData(startTime, endTime)</b>
Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>sensor→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>sensor→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>sensor→get_unit()</b>
Retourne l'unité dans laquelle la mesure est exprimée.
<b>sensor→get_userData()</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>sensor→isOnline()</b>
Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.
<b>sensor→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.
<b>sensor→load(msValidity)</b>
Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.
<b>sensor→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>sensor→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.
<b>sensor→nextSensor()</b>
Continue l'énumération des senseurs commencée à l'aide de yFirstSensor( ).
<b>sensor→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>sensor→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>sensor→set_highestValue(newval)</b>
Modifie la mémoire de valeur maximale observée.
<b>sensor→set_logFrequency(newval)</b>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**sensor→set\_logicalName(newval)**

Modifie le nom logique du senseur.

**sensor→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**sensor→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**sensor→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**sensor→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**sensor→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YSensor.FindSensor()****YSensor****yFindSensor()yFindSensor( )**

Permet de retrouver un senseur d'après un identifiant donné.

**YSensor\* yFindSensor( string func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le senseur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSensor.isOnline()` pour tester si le senseur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le senseur sans ambiguïté

**Retourne :**

un objet de classe `YSensor` qui permet ensuite de contrôler le senseur.

## **YSensor.FirstSensor() yFirstSensor()yFirstSensor( )**

**YSensor**

Commence l'énumération des senseurs accessibles par la librairie.

**YSensor\* yFirstSensor( )**

Utiliser la fonction `YSensor.nextSensor()` pour itérer sur les autres senseurs.

**Retourne :**

un pointeur sur un objet `YSensor`, correspondant au premier senseur accessible en ligne, ou `null` si il n'y a pas de senseurs disponibles.

---

<b>sensor→calibrateFromPoints()</b> <b>sensor→calibrateFromPoints( )</b>	<b>YSensor</b>
-----------------------------------------------------------------------------	----------------

---

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor->describe()****YSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le senseur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**sensor→get\_advertisedValue()**  
**sensor→advertisedValue()sensor→**  
**get\_advertisedValue( )**

**YSensor**

Retourne la valeur courante du senseur (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du senseur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**sensor→get\_currentRawValue()**  
**sensor→currentRawValue()sensor→**  
**get\_currentRawValue( )**

---

**YSensor**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

double **get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

---

```
sensor->get_currentValue()  
sensor->currentValue()sensor->  
get_currentValue()
```

YSensor

Retourne la valeur actuelle de la mesure.

```
double get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la mesure

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**sensor→get\_errorMessage()** YSensor  
**sensor→errorMessage()sensor→**  
**get\_errorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

**string 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 senseur.

---

**sensor→get\_errorType()****YSensor****sensor→errorType()sensor→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

**sensor→get\_friendlyName()** YSensor  
**sensor→friendlyName()** **sensor→get\_friendlyName( )**

---

Retourne un identifiant global du senseur au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du senseur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du senseur (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le senseur en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**sensor→get\_functionDescriptor()**  
**sensor→functionDescriptor()sensor→**  
**get\_functionDescriptor( )**

**YSensor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**sensor→get\_functionId()**

**YSensor**

**sensor→functionId()sensor→get\_functionId()**

---

Retourne l'identifiant matériel du senseur, sans référence au module.

string **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

---

**sensor→get\_hardwareId()****YSensor****sensor→hardwareId()sensor→get\_hardwareId( )**

---

Retourne l'identifiant matériel unique du senseur au format SERIAL.FUNCTIONID.**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du senseur (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**sensor→get\_highestValue()**  
**sensor→highestValue()****sensor→get\_highestValue( )**

**YSensor**

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

double **get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

**sensor→get\_logFrequency()**  
**sensor→logFrequency()sensor→**  
**get\_logFrequency( )**

**YSensor**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**sensor→get\_logicalName()** YSensor  
**sensor→logicalName()** **sensor→get\_logicalName( )**

---

Retourne le nom logique du senseur.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du senseur. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**sensor→get\_lowestValue()**  
**sensor→lowestValue()**sensor→  
**get\_lowestValue( )**

**YSensor**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

double **get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**sensor→get\_module()**

**YSensor**

**sensor→module()sensor→get\_module( )**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**sensor→get\_recordedData()**  
**sensor→recordedData()sensor→**  
**get\_recordedData()**

**YSensor**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

**sensor→get\_reportFrequency()** YSensor  
**sensor→reportFrequency()** **sensor→get\_reportFrequency( )**

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

<b>sensor→get_resolution()</b>	<b>YSensor</b>
<b>sensor→resolution()sensor→get_resolution()</b>	

---

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**sensor→get\_unit()**

**YSensor**

**sensor→unit()sensor→get\_unit()**

---

Retourne l'unité dans laquelle la mesure est exprimée.

string **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

**sensor→get(userData)****YSensor****sensor→userData()sensor→get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**sensor->isOnline()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du senseur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le senseur est joignable, false sinon

**sensor→load()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→loadCalibrationPoints()** **sensor→loadCalibrationPoints()**

**YSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→nextSensor()****YSensor**

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

`YSensor * nextSensor()`

**Retourne :**

un pointeur sur un objet `YSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**sensor→registerTimedReportCallback()** **sensor→registerTimedReportCallback( )**

**YSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YSensorTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

---

<b>sensor-&gt;registerValueCallback()</b> <b>sensor-&gt; registerValueCallback( )</b>	<b>YSensor</b>
----------------------------------------------------------------------------------------------	----------------

---

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YSensorValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**sensor→set\_highestValue()** YSensor  
**sensor→setHighestValue()** **sensor→set\_highestValue( )**

---

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_logFrequency()**  
**sensor→setLogFrequency()**sensor→  
**set\_logFrequency( )**

**YSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>sensor→set_logicalName()</b>	<b>YSensor</b>
<b>sensor→setLogicalName()</b> <b>sensor→</b>	
<b>set_logicalName( )</b>	

---

Modifie le nom logique du senseur.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du senseur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_lowestValue()**  
**sensor→setLowestValue()****sensor→set\_lowestValue( )**

**YSensor**

Modifie la mémoire de valeur minimale observée.

int **set\_lowestValue( double newval)**

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_reportFrequency()** YSensor  
**sensor→setReportFrequency() sensor→**  
**set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

```
sensor->set_resolution()  
sensor->setResolution()sensor->  
set_resolution( )
```

**YSensor**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set(userData)**

**YSensor**

**sensor→setUserData()sensor→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.37. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'intervalle de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_servo.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YServo = yoctolib.YServo;
php	require_once('yocto_servo.php');
cpp	#include "yocto_servo.h"
m	#import "yocto_servo.h"
pas	uses yocto_servo;
vb	yocto_servo.vb
cs	yocto_servo.cs
java	import com.yoctopuce.YoctoAPI.YServo;
py	from yocto_servo import *

### Fonction globales

#### yFindServo(func)

Permet de retrouver un servo d'après un identifiant donné.

#### yFirstServo()

Commence l'énumération des servo accessibles par la librairie.

### Méthodes des objets YServo

#### servo->describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format TYPE (NAME) = SERIAL.FUNCTIONID.

#### servo->get\_advertisedValue()

Retourne la valeur courante du servo (pas plus de 6 caractères).

#### servo->get\_enabled()

Retourne l'état de fonctionnement du \$FUNCTION\$.

#### servo->get\_enabledAtPowerOn()

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

#### servo->get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

#### servo->get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

#### servo->get\_friendlyName()

Retourne un identifiant global du servo au format NOM\_MODULE . NOM\_FONCTION.

#### servo->get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### servo->get\_functionId()

Retourne l'identifiant matériel du servo, sans référence au module.

#### servo->get\_hardwareId()

Retourne l'identifiant matériel unique du servo au format SERIAL.FUNCTIONID.

#### servo->get\_logicalName()

Retourne le nom logique du servo.

### 3. Reference

<b>servo→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>servo→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>servo→get_neutral()</b>	Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.
<b>servo→get_position()</b>	Retourne la position courante du servo.
<b>servo→get_positionAtPowerOn()</b>	Retourne la position du servo au démarrage du module.
<b>servo→get_range()</b>	Retourne la plage d'utilisation du servo.
<b>servo→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>servo→isOnline()</b>	Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.
<b>servo→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.
<b>servo→load(msValidity)</b>	Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.
<b>servo→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.
<b>servo→move(target, ms_duration)</b>	Déclenche un mouvement à vitesse constante vers une position donnée.
<b>servo→nextServo()</b>	Continue l'énumération des servo commencée à l'aide de yFirstServo( ).
<b>servo→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>servo→set_enabled(newval)</b>	Démarre ou arrête le \$FUNCTION\$.
<b>servo→set_enabledAtPowerOn(newval)</b>	Configure l'état du générateur de signal de commande du servo au démarrage du module.
<b>servo→set_logicalName(newval)</b>	Modifie le nom logique du servo.
<b>servo→set_neutral(newval)</b>	Modifie la durée de l'impulsion correspondant à la position neutre du servo.
<b>servo→set_position(newval)</b>	Modifie immédiatement la consigne de position du servo.
<b>servo→set_positionAtPowerOn(newval)</b>	Configure la position du servo au démarrage du module.
<b>servo→set_range(newval)</b>	Modifie la plage d'utilisation du servo, en pourcents.
<b>servo→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>servo→wait_async(callback, context)</b>	

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YServo.FindServo()****YServo****yFindServo()yFindServo( )**

Permet de retrouver un servo d'après un identifiant donné.

**YServo\* yFindServo( const string& 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()****YServo****yFirstServo()yFirstServo( )**

Commence l'énumération des servo accessibles par la librairie.

**YServo\* yFirstServo( )**

Utiliser la fonction `YServo.nextServo()` pour itérer sur les autres servo.

**Retourne :**

un pointeur sur un objet `YServo`, correspondant au premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

**servo→describe()****YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le servo (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**servo→get\_advertisedValue()**  
**servo→advertisedValue()servo→**  
**get\_advertisedValue( )**

**YServo**

Retourne la valeur courante du servo (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du servo (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**servo→get\_enabled()**

**YServo**

**servo→enabled()servo→get\_enabled()**

---

Retourne l'état de fonctionnement du \$FUNCTION\$.

**Y\_ENABLED\_enum get\_enabled( )**

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du \$FUNCTION\$

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

---

<b>servo→get_enabledAtPowerOn()</b>	<b>YServo</b>
<b>servo→enabledAtPowerOn()servo→</b>	
<b>get_enabledAtPowerOn( )</b>	

---

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

**Y\_ENABLEDATPOWERON\_enum get\_enabledAtPowerOn( )**

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

**servo→getErrorMessage()**  
**servo→errorMessage()**servo→  
**getErrorMessage( )**

---

**YServo**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du servo.

---

**servo→get\_errorType()****YServo****servo→errorType()servo→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du servo.

**servo→get\_friendlyName()**  
**servo→friendlyName()** servo→  
**get\_friendlyName( )**

---

**YServo**

Retourne un identifiant global du servo au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du servo si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du servo (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le servo en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**servo→get\_functionDescriptor()**  
**servo→functionDescriptor()servo→**  
**get\_functionDescriptor( )**

**YServo**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**servo→get\_functionId()**

**YServo**

**servo→functionId()servo→get\_functionId()**

---

Retourne l'identifiant matériel du servo, sans référence au module.

string **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le servo (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**servo→get\_hardwareId()****YServo****servo→hardwareId()servo→get\_hardwareId( )**

Retourne l'identifiant matériel unique du servo au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du servo (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le servo (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**servo→get\_logicalName()**

**YServo**

**servo→logicalName()servo→get\_logicalName( )**

---

Retourne le nom logique du servo.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du servo. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**servo→get\_module()****YServo****servo→module()servo→get\_module( )**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

**servo→get\_neutral()**

**YServo**

**servo→neutral()servo→get\_neutral( )**

---

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

**int get\_neutral( )**

**Retourne :**

un entier représentant la durée en microsecondes de l'impulsion correspondant au neutre du servo

En cas d'erreur, déclenche une exception ou retourne Y\_NEUTRAL\_INVALID.

**servo→get\_position()****YServo****servo→position()servo→get\_position()**

Retourne la position courante du servo.

```
int get_position( )
```

**Retourne :**

un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne Y\_POSITION\_INVALID.

**servo→get\_positionAtPowerOn()**  
**servo→positionAtPowerOn()servo→**  
**get\_positionAtPowerOn( )**

---

**YServo**

Retourne la position du servo au démarrage du module.

**int get\_positionAtPowerOn( )**

**Retourne :**

un entier représentant la position du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_POSITIONATPOWERON\_INVALID.

---

**servo→get\_range()****YServo****servo→range()servo→get\_range( )**

---

Retourne la plage d'utilisation du servo.

```
int get_range( )
```

**Retourne :**

un entier représentant la plage d'utilisation du servo

En cas d'erreur, déclenche une exception ou retourne Y\_RANGE\_INVALID.

**servo→get(userData())**

**YServo**

**servo→userData()servo→get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * get(userData())
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**servo→isOnline()****YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du servo sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le servo est joignable, false sinon

**servo→load()servo→load( )****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→move()****servo→move( )****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

```
int move( int target, int ms_duration)
```

**Paramètres :**

**target** nouvelle position à la fin du mouvement

**ms\_duration** durée totale du mouvement, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→nextServo()** **servo→nextServo( )**

**YServo**

---

Continue l'énumération des servo commencée à l'aide de `yFirstServo( )`.

`YServo * nextServo( )`

**Retourne :**

un pointeur sur un objet `YServo` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

<b>servo→registerValueCallback()</b> servo→ <b>registerValueCallback( )</b>	<b>YServo</b>
-----------------------------------------------------------------------------------	---------------

---

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YServoValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**servo→set\_enabled()**

**YServo**

**servo→setEnabled()servo→set\_enabled( )**

---

Démarre ou arrête le \$FUNCTION\$.

```
int set_enabled( Y_ENABLED_enum newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_enabledAtPowerOn()****YServo****servo→setEnabledAtPowerOn()servo→****set\_enabledAtPowerOn( )**

Configure l'état du générateur de signal de commande du servo au démarrage du module.

```
int set_enabledAtPowerOn( Y_ENABLEDATPOWERON_enum newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>servo→set_logicalName()</b>	<b>YServo</b>
<b>servo→setLogicalName()</b> servo→ <b>set_logicalName( )</b>	

---

Modifie le nom logique du servo.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du servo.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_neutral()****YServo****servo→setNeutral()servo→set\_neutral( )**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
int set_neutral( int newval)
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_position()** **YServo**  
**servo→setPosition()****servo→set\_position( )**

---

Modifie immédiatement la consigne de position du servo.

**int set\_position( int newval)**

**Paramètres :**

**newval** un entier représentant immédiatement la consigne de position du servo

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_positionAtPowerOn()****YServo****servo→setPositionAtPowerOn()servo→****set\_positionAtPowerOn( )**

Configure la position du servo au démarrage du module.

**int set\_positionAtPowerOn( int newval)**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :****newval** un entier**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_range()****YServo****servo→setRange()servo→set\_range( )**

Modifie la plage d'utilisation du servo, en pourcents.

**int set\_range( int newval)**

La valeur 100% correspond à un signal de commande standard, variant de 1 [ms] à 2 [ms]. Pour les servos supportent une plage double, de 0.5 [ms] à 2.5 [ms], vous pouvez utiliser une valeur allant jusqu'à 200%. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la plage d'utilisation du servo, en pourcents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set(userData)****YServo****servo→setUserData()servo→set(userData( )**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.38. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_temperature.js'></script>
nodejs var yoctolib = require('yoctolib');
var YTemperature = yoctolib.YTemperature;
php require_once('yocto_temperature.php');
cpp #include "yocto_temperature.h"
m #import "yocto_temperature.h"
pas uses yocto_temperature;
vb yocto_temperature.vb
cs yocto_temperature.cs
java import com.yoctopuce.YoctoAPI.YTemperature;
py from yocto_temperature import *

```

### Fonction globales

#### yFindTemperature(func)

Permet de retrouver un capteur de température d'après un identifiant donné.

#### yFirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

### Méthodes des objets YTemperature

#### temperature→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### temperature→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### temperature→get\_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

#### temperature→get\_currentRawValue()

Retourne la valeur brute renournée par le capteur (sans arrondi ni calibration).

#### temperature→get\_currentValue()

Retourne la valeur actuelle de la température.

#### temperature→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

#### temperature→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

#### temperature→get\_friendlyName()

Retourne un identifiant global du capteur de température au format NOM\_MODULE . NOM\_FONCTION.

#### temperature→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### temperature→get\_functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

#### temperature→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format SERIAL.FUNCTIONID.
<b>temperature→get_highestValue()</b>
Retourne la valeur maximale observée pour la température depuis le démarrage du module.
<b>temperature→get_logFrequency()</b>
Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>temperature→get_logicalName()</b>
Retourne le nom logique du capteur de température.
<b>temperature→get_lowestValue()</b>
Retourne la valeur minimale observée pour la température depuis le démarrage du module.
<b>temperature→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>temperature→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>temperature→get_recordedData(startTime, endTime)</b>
Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>temperature→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>temperature→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>temperature→get_sensorType()</b>
Retourne le type de capteur de température utilisé par le module
<b>temperature→get_unit()</b>
Retourne l'unité dans laquelle la température est exprimée.
<b>temperature→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>temperature→isOnline()</b>
Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.
<b>temperature→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.
<b>temperature→load(msValidity)</b>
Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.
<b>temperature→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>temperature→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.
<b>temperature→nextTemperature()</b>
Continue l'énumération des capteurs de température commencée à l'aide de yFirstTemperature( ).
<b>temperature→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>temperature→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>temperature→set_highestValue(newval)</b>

### 3. Reference

---

Modifie la mémoire de valeur maximale observée.

**temperature→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**temperature→set\_logicalName(newval)**

Modifie le nom logique du capteur de température.

**temperature→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**temperature→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**temperature→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**temperature→set\_sensorType(newval)**

Change le type de senseur utilisé par le module.

**temperature→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**temperature→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YTemperature.FindTemperature()****YTemperature****yFindTemperature()yFindTemperature( )**

Permet de retrouver un capteur de température d'après un identifiant donné.

**YTemperature\* yFindTemperature( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de température soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de température sans ambiguïté

**Retourne :**

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

## **YTemperature.FirstTemperature()**

**YTemperature**

### **yFirstTemperature()yFirstTemperature( )**

Commence l'énumération des capteurs de température accessibles par la librairie.

**YTemperature\* yFirstTemperature( )**

Utiliser la fonction `YTemperature.nextTemperature( )` pour itérer sur les autres capteurs de température.

**Retourne :**

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

## temperature→calibrateFromPoints()temperature→ calibrateFromPoints( )

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

### 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()****YTemperature**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de température (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**temperature→get\_advertisedValue()**

**YTemperature**

**temperature→advertisedValue()temperature→  
get\_advertisedValue( )**

---

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**temperature→get\_currentRawValue()**

**YTemperature**

**temperature→currentRawValue()** **temperature→get\_currentRawValue( )**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

**double get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

**temperature→get\_currentValue()**

**YTemperature**

**temperature→currentValue()temperature→  
get\_currentValue()**

Retourne la valeur actuelle de la température.

**double get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la valeur actuelle de la température

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**temperature→getErrorMessage()**

**YTemperature**

**temperature→errorMessage()temperature→  
getErrorMessage( )**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

**temperature→get\_errorType()**

**YTemperature**

**temperature→errorType()temperature→**

**get\_errorType( )**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

**temperature→get\_friendlyName()** YTemperature  
**temperature→friendlyName()temperature→**  
**get\_friendlyName( )**

Retourne un identifiant global du capteur de température au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de température si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de température (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de température en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**temperature→get\_functionDescriptor()**

**YTemperature**

**temperature→functionDescriptor()temperature→get\_functionDescriptor( )**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**temperature→get\_functionId()** YTemperature  
**temperature→functionId()** **temperature→get\_functionId( )**

---

Retourne l'identifiant matériel du capteur de température, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de température (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**temperature→get\_hardwareId()****YTemperature****temperature→hardwareId()temperature→  
get\_hardwareId( )**

Retourne l'identifiant matériel unique du capteur de température au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de température (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**temperature→get\_highestValue()** YTemperature  
**temperature→highestValue()temperature→**  
**get\_highestValue( )**

---

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

```
double get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**temperature→get\_logFrequency()**

**YTemperature**

**temperature→logFrequency()temperature→**

**get\_logFrequency( )**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**temperature→get\_logicalName()** YTemperature  
**temperature→logicalName()temperature→**  
**get\_logicalName( )**

---

Retourne le nom logique du capteur de température.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de température. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**temperature→get\_lowestValue()**

**YTemperature**

**temperature→lowestValue()temperature→  
get\_lowestValue( )**

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

**double get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**temperature→get\_module()**  
**temperature→module()temperature→**  
**get\_module( )**

---

**YTemperature**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**temperature→get\_recordedData()****YTemperature****temperature→recordedData()temperature→  
get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**temperature→get\_reportFrequency()** YTemperature  
**temperature→reportFrequency()** **temperature→get\_reportFrequency( )**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

<b>temperature→get_resolution()</b>	<b>YTemperature</b>
<b>temperature→resolution()temperature→get_resolution( )</b>	

---

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**temperature→get\_sensorType()** YTemperature  
**temperature→sensorType()** **temperature→get\_sensorType( )**

Retourne le type de capteur de température utilisé par le module

**Y\_SENSORTYPE\_enum get\_sensorType( )**

**Retourne :**

une valeur parmi Y\_SENSORTYPE\_DIGITAL, Y\_SENSORTYPE\_TYPE\_K,  
Y\_SENSORTYPE\_TYPE\_E, Y\_SENSORTYPE\_TYPE\_J, Y\_SENSORTYPE\_TYPE\_N,  
Y\_SENSORTYPE\_TYPE\_R, Y\_SENSORTYPE\_TYPE\_S, Y\_SENSORTYPE\_TYPE\_T,  
Y\_SENSORTYPE\_PT100\_4WIRES, Y\_SENSORTYPE\_PT100\_3WIRES et  
Y\_SENSORTYPE\_PT100\_2WIRES représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_SENSORTYPE\_INVALID.

**temperature→get\_unit()****YTemperature****temperature→unit()temperature→get\_unit( )**

Retourne l'unité dans laquelle la température est exprimée.

**string get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**temperature→get(userData)**  
**temperature→userData()temperature→**  
**get(userData)**

**YTemperature**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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()****YTemperature**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de température est joignable, false sinon

**temperature→load()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→loadCalibrationPoints(temperature→  
loadCalibrationPoints( )****YTemperature**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→nextTemperature()** **temperature→** **YTemperature**  
**nextTemperature( )**

---

Continue l'énumération des capteurs de température commencée à l'aide de **yFirstTemperature( )**.

**YTemperature \* nextTemperature( )**

**Retourne :**

un pointeur sur un objet **YTemperature** accessible en ligne, ou **null** lorsque l'énumération est terminée.

**temperature→registerTimedReportCallback()****YTemperature****temperature→registerTimedReportCallback( )**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YTemperatureTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**temperature**→**registerValueCallback()**  
**temperature**→**registerValueCallback( )**

**YTemperature**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**int registerValueCallback( YTemperatureValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**temperature→set\_highestValue()** **YTemperature**  
**temperature→setHighestValue()** **temperature→**  
**set\_highestValue()**

---

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logFrequency()** YTemperature  
**temperature→setLogFrequency()** **temperature→set\_logFrequency( )**

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logicalName()**

**YTemperature**

**temperature→setLogicalName()temperature→  
set\_logicalName( )**

---

Modifie le nom logique du capteur de température.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de température.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_lowestValue()** YTemperature  
**temperature→setLowestValue()** **temperature→set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_reportFrequency()****YTemperature****temperature→setReportFrequency()temperature→  
set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_resolution()** YTemperature  
**temperature→setResolution()** **temperature→set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

**int set\_resolution( double newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>temperature→set_sensorType()</b>	<b>YTemperature</b>
<b>temperature→setSensorType()temperature→set_sensorType( )</b>	

Change le type de senseur utilisé par le module.

```
int set_sensorType( Y_SENSORTYPE_enum newval)
```

Cette fonction sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital. N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi Y\_SENSORTYPE\_DIGITAL, Y\_SENSORTYPE\_TYPE\_K, Y\_SENSORTYPE\_TYPE\_E, Y\_SENSORTYPE\_TYPE\_J, Y\_SENSORTYPE\_TYPE\_N, Y\_SENSORTYPE\_TYPE\_R, Y\_SENSORTYPE\_TYPE\_S, Y\_SENSORTYPE\_TYPE\_T, Y\_SENSORTYPE\_PT100\_4WIRES, Y\_SENSORTYPE\_PT100\_3WIRES et Y\_SENSORTYPE\_PT100\_2WIRES

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set(userData)** YTemperature  
**temperature→setUserData()** **temperature→set(userData())**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.39. Interface de la fonction Tilt

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_tilt.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YTilt = yoctolib.YTilt;
php	require_once('yocto_tilt.php');
cpp	#include "yocto_tilt.h"
m	#import "yocto_tilt.h"
pas	uses yocto_tilt;
vb	yocto_tilt.vb
cs	yocto_tilt.cs
java	import com.yoctopuce.YoctoAPI.YTilt;
py	from yocto_tilt import *

### Fonction globales

#### yFindTilt(func)

Permet de retrouver un inclinomètre d'après un identifiant donné.

#### yFirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

### Méthodes des objets YTilt

#### tilt→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### tilt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### tilt→get\_advertisedValue()

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

#### tilt→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### tilt→get\_currentValue()

Retourne la valeur actuelle de l'inclinaison.

#### tilt→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

#### tilt→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

#### tilt→get\_friendlyName()

Retourne un identifiant global de l'inclinomètre au format NOM\_MODULE . NOM\_FONCTION.

#### tilt→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### tilt→get\_functionId()

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

#### tilt→get\_hardwareId()

Retourne l'identifiant matériel unique de l'inclinomètre au format SERIAL . FUNCTIONID.

### 3. Reference

<b>tilt→get_highestValue()</b>	Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.
<b>tilt→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>tilt→get_logicalName()</b>	Retourne le nom logique de l'inclinomètre.
<b>tilt→get_lowestValue()</b>	Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.
<b>tilt→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>tilt→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>tilt→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>tilt→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>tilt→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>tilt→get_unit()</b>	Retourne l'unité dans laquelle l'inclinaison est exprimée.
<b>tilt→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>tilt→isOnline()</b>	Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.
<b>tilt→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.
<b>tilt→load(msValidity)</b>	Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.
<b>tilt→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>tilt→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.
<b>tilt→nextTilt()</b>	Continue l'énumération des inclinomètres commencée à l'aide de yFirstTilt( ).
<b>tilt→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>tilt→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>tilt→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.
<b>tilt→set_logFrequency(newval)</b>	Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**tilt→set\_logicalName(newval)**

Modifie le nom logique de l'inclinomètre.

**tilt→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**tilt→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**tilt→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**tilt→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**tilt→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YTilt.FindTilt()****YTilt****yFindTilt()yFindTilt( )**

Permet de retrouver un inclinomètre d'après un identifiant donné.

**YTilt\* yFindTilt( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'inclinomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTilt.isOnline()` pour tester si l'inclinomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'inclinomètre sans ambiguïté

**Retourne :**

un objet de classe `YTilt` qui permet ensuite de contrôler l'inclinomètre.

**YTilt.FirstTilt()****YTilt****yFirstTilt()yFirstTilt( )**

Commence l'énumération des inclinomètres accessibles par la librairie.

**YTilt\* yFirstTilt( )**

Utiliser la fonction YTilt.nextTilt( ) pour itérer sur les autres inclinomètres.

**Retourne :**

un pointeur sur un objet YTilt, correspondant au premier inclinomètre accessible en ligne, ou null si il n'y a pas de inclinomètres disponibles.

**tilt→calibrateFromPoints()tilt→  
calibrateFromPoints( )**

YTilt

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→describe()****YTilt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant l'inclinomètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**tilt→get\_advertisedValue()**  
**tilt→advertisedValue()**tilt→  
**get\_advertisedValue( )**

---

YTilt

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

string **get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**tilt→get\_currentRawValue()**  
**tilt→currentRawValue()**  
**tilt→get\_currentRawValue( )**

YTilt

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

**double get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**tilt→get\_currentValue()** YTilt  
**tilt→currentValue()tilt→get\_currentValue()**

---

Retourne la valeur actuelle de l'inclinaison.

```
double get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'inclinaison

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**tilt→get\_errorMessage()****YTilt****tilt→errorMessage()tilt→get\_errorMessage( )**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
string get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

<b>tilt→get_errorType()</b>	<b>YTilt</b>
<b>tilt→errorType()tilt→get_errorType( )</b>	

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

**tilt→get\_friendlyName()**

YTilt

**tilt→friendlyName()tilt→get\_friendlyName( )**

Retourne un identifiant global de l'inclinomètre au format NOM\_MODULE . NOM\_FONCTION.

```
string get_friendlyName( )
```

Le chaîne renvoyée utilise soit les noms logiques du module et de l'inclinomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'inclinomètre (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

`tilt->get_functionDescriptor()`  
`tilt->functionDescriptor() tilt->`  
`get_functionDescriptor( )`

---

YTilt

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**tilt→get\_functionId()****YTilt****tilt→functionId()tilt→get\_functionId()**

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

<b>tilt→get_hardwareId()</b>	<b>YTilt</b>
<b>tilt→hardwareId()tilt→get_hardwareId( )</b>	

---

Retourne l'identifiant matériel unique de l'inclinomètre au format SERIAL.FUNCTIONID.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'inclinomètre (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**tilt→get\_highestValue()****YTilt****tilt→highestValue()tilt→get\_highestValue( )**

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

```
double get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

---

<b>tilt→get_logFrequency()</b>	<b>YTilt</b>
<b>tilt→logFrequency()</b>	<b>tilt→get_logFrequency( )</b>

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
string get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

**tilt→get\_logicalName()****YTilt****tilt→logicalName()tilt→get\_logicalName( )**

Retourne le nom logique de l'inclinomètre.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'inclinomètre. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**tilt→get\_lowestValue()** YTilt  
**tilt→lowestValue()tilt→get\_lowestValue( )**

---

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

**double get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**tilt→get\_module()****YTilt****tilt→module()tilt→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* 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`

---

<b>tilt→get_recordedData()</b>	<b>YTilt</b>
<b>tilt→recordedData()tilt→get_recordedData( )</b>	

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**tilt→get\_reportFrequency()**

YTilt

**tilt→reportFrequency()tilt→****get\_reportFrequency( )**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
string get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**tilt→get\_resolution()** YTilt  
**tilt→resolution()tilt→get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**tilt→get\_unit()****YTilt****tilt→unit()tilt→get\_unit()**

Retourne l'unité dans laquelle l'inclinaison est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'inclinaison est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**tilt→get(userData)**

YTilt

**tilt→userData()tilt→get(userData()**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**tilt→isOnline()**

YTilt

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache de l'inclinomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'inclinomètre est joignable, `false` sinon

**tilt→load()****YTilt**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→loadCalibrationPoints()tilt→  
loadCalibrationPoints()****YTilt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **tilt→nextTilt()tilt→nextTilt()**

**YTilt**

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

`YTilt * nextTilt( )`

**Retourne :**

un pointeur sur un objet `YTilt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

**tilt→registerTimedReportCallback()tilt→  
registerTimedReportCallback( )****YTilt**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YTiltTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**tilt→registerValueCallback()** **tilt→registerValueCallback( )**

**YTilt**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**int registerValueCallback( YTiltValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**tilt→set\_highestValue()**

YTilt

**tilt→setHighestValue()tilt→set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_logFrequency()**  
**tilt→setLogFrequency()** **tilt→**  
**set\_logFrequency( )**

YTilt

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_logicalName()**

YTilt

**tilt→setLogicalName()tilt→set\_logicalName( )**

Modifie le nom logique de l'inclinomètre.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'inclinomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_lowestValue()** YTilt  
**tilt→setLowestValue()** YTilt

---

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_reportFrequency()**  
**tilt→setReportFrequency()** **tilt→**  
**set\_reportFrequency( )**

YTilt

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_resolution()** YTilt  
**tilt→setResolution()** YTilt

---

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set(userData)**

YTilt

**tilt→setUserData()|tilt→set(userData())**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* 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

## 3.40. Interface de la fonction Voc

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_voc.js'></script>
nodejs var yoctolib = require('yoctolib');
var YVoc = yoctolib.YVoc;
php require_once('yocto_voc.php');
cpp #include "yocto_voc.h"
m #import "yocto_voc.h"
pas uses yocto_voc;
vb yocto_voc.vb
cs yocto_voc.cs
java import com.yoctopuce.YoctoAPI.YVoc;
py from yocto_voc import *

```

### Fonction globales

#### yFindVoc(func)

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

#### yFirstVoc()

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

### Méthodes des objets YVoc

#### voc→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### voc→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### voc→get\_advertisedValue()

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

#### voc→get\_currentRawValue()

Retourne la valeur brute renournée par le capteur (sans arrondi ni calibration).

#### voc→get\_currentValue()

Retourne la mesure actuelle du taux de VOC estimé.

#### voc→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

#### voc→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

#### voc→get\_friendlyName()

Retourne un identifiant global du capteur de Composés Organiques Volatils au format NOM\_MODULE . NOM\_FONCTION.

#### voc→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### voc→get\_functionId()

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

<b>voc→get.hardwareId()</b>	Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.
<b>voc→get_highestValue()</b>	Retourne la valeur maximale observée pour le taux de VOC estimé.
<b>voc→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>voc→get_logicalName()</b>	Retourne le nom logique du capteur de Composés Organiques Volatils.
<b>voc→get_lowestValue()</b>	Retourne la valeur minimale observée pour le taux de VOC estimé.
<b>voc→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>voc→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>voc→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>voc→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>voc→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>voc→get_unit()</b>	Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.
<b>voc→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>voc→isOnline()</b>	Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.
<b>voc→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.
<b>voc→load(msValidity)</b>	Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.
<b>voc→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>voc→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.
<b>voc→nextVoc()</b>	Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de yFirstVoc( ).
<b>voc→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.

### 3. Reference

#### **voc→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **voc→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

#### **voc→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### **voc→set\_logicalName(newval)**

Modifie le nom logique du capteur de Composés Organiques Volatils.

#### **voc→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

#### **voc→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **voc→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **voc→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **voc→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YVoc.FindVoc()****YVoc****yFindVoc()yFindVoc( )**

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

**YVoc\* yFindVoc( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de Composés Organiques Volatils soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode YVoc.isOnline() pour tester si le capteur de Composés Organiques Volatils est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de Composés Organiques Volatils sans ambiguïté

**Retourne :**

un objet de classe YVoc qui permet ensuite de contrôler le capteur de Composés Organiques Volatils.

## YVoc.FirstVoc()

**YVoc**

### yFirstVoc()yFirstVoc( )

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

**YVoc\* yFirstVoc( )**

Utiliser la fonction YVoc.nextVoc( ) pour itérer sur les autres capteurs de Composés Organiques Volatils.

**Retourne :**

un pointeur sur un objet YVoc, correspondant au premier capteur de Composés Organiques Volatils accessible en ligne, ou null si il n'y a pas de capteurs de Composés Organiques Volatils disponibles.

**voc→calibrateFromPoints()voc→  
calibrateFromPoints( )**

**YVoc**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc->describe()****YVoc**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format TYPE ( NAME ) =SERIAL . FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**voc→get\_advertisedValue()**

**YVoc**

**voc→advertisedValue()voc→**

**get\_advertisedValue( )**

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

```
string get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**voc→get\_currentRawValue()**  
**voc→currentRawValue()voc→**  
**get\_currentRawValue( )**

---

**YVoc**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

double **get\_currentRawValue( )**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**voc→get\_currentValue()**

**YVoc**

**voc→currentValue()voc→get\_currentValue( )**

Retourne la mesure actuelle du taux de VOC estimé.

double **get\_currentValue( )**

**Retourne :**

une valeur numérique représentant la mesure actuelle du taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**voc→get\_errorMessage()**

**YVoc**

**voc→errorMessage()voc→get\_errorMessage( )**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

---

**voc→get\_errorType()****YVoc****voc→errorType()voc→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

**voc→get\_friendlyName()****YVoc****voc→friendlyName()voc→get\_friendlyName( )**

Retourne un identifiant global du capteur de Composés Organiques Volatils au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de Composés Organiques Volatils si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**voc→get\_functionDescriptor()**  
**voc→functionDescriptor()voc→**  
**get\_functionDescriptor( )**

**YVoc**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voc→get\_functionId()**

**YVoc**

**voc→functionId()voc→get\_functionId()**

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voc→get.hardwareId()****YVoc****voc→hardwareId()voc→get.hardwareId( )**

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.

**string get.hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**voc→get\_highestValue()**

**YVoc**

**voc→highestValue()voc→get\_highestValue()**

---

Retourne la valeur maximale observée pour le taux de VOC estimé.

double **get\_highestValue( )**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

**voc→get\_logFrequency()****YVoc****voc→logFrequency()voc→get\_logFrequency( )**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**string get\_logFrequency( )****Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**voc→get\_logicalName()**

**YVoc**

**voc→logicalName()voc→get\_logicalName( )**

---

Retourne le nom logique du capteur de Composés Organiques Volatils.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**voc→get\_lowestValue()**

**YVoc**

**voc→lowestValue()voc→get\_lowestValue( )**

Retourne la valeur minimale observée pour le taux de VOC estimé.

double **get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**voc→get\_module()**

**YVoc**

**voc→module()voc→get\_module( )**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**voc→get\_recordedData()****YVoc****voc→recordedData()voc→get\_recordedData( )**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voc→get\_reportFrequency()**  
**voc→reportFrequency()voc→**  
**get\_reportFrequency( )**

**YVoc**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

string **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**voc→get\_resolution()****YVoc****voc→resolution()voc→get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**voc→get\_unit()**

**YVoc**

**voc→unit()voc→get\_unit()**

---

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

string **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le taux de VOC estimé est exprimée

En cas d'erreur, déclenche une exception ou retourne **Y\_UNIT\_INVALID**.

**voc→get(userData)****YVoc****voc→userData()voc→get(userData)()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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.

**voc→isOnline()****YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du capteur de Composés Organiques Volatils sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de Composés Organiques Volatils est joignable, false sinon

**voc→load()voc→load( )****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→loadCalibrationPoints()** **voc→  
loadCalibrationPoints()**

**YVoc**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→nextVoc()voc→nextVoc( )****YVoc**

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

**YVoc \* nextVoc( )****Retourne :**

un pointeur sur un objet YVOC accessible en ligne, ou `null` lorsque l'énumération est terminée.

**voc→registerTimedReportCallback()**  
voc→  
**registerTimedReportCallback( )**

**YVoc**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**int registerTimedReportCallback( YVocTimedReportCallback **callback**)**

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**voc→registerValueCallback()voc→  
registerValueCallback( )****YVoc**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YVocValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**voc→set\_highestValue()**

**YVoc**

**voc→setHighestValue()|voc→set\_highestValue( )**

---

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour le taux de VOC estimé

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_logFrequency()****YVoc****voc→setLogFrequency()voc→set\_logFrequency( )**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( const string& newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_logicalName()** YVoc  
**voc→setLogicalName()voc→set\_logicalName( )**

---

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_lowestValue()****YVoc****voc→setLowestValue()voc→set\_lowestValue( )**

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour le taux de VOC estimé

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_reportFrequency()**  
**voc→setReportFrequency()voc→**  
**set\_reportFrequency( )**

**YVoc**

Modifie la fréquence de notification périodique des valeurs mesurées.

**int set\_reportFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_resolution()****YVoc****voc→setResolution()voc→set\_resolution( )**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set(userData)**  
**voc→setUserData()voc→set(userData)****YVoc**

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).

```
void set(userData void* 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

## 3.41. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_voltage.js'></script>
nodejs var yoctolib = require('yoctolib');
var YVoltage = yoctolib.YVoltage;
php require_once('yocto_voltage.php');
cpp #include "yocto_voltage.h"
m #import "yocto_voltage.h"
pas uses yocto_voltage;
vb yocto_voltage.vb
cs yocto_voltage.cs
java import com.yoctopuce.YoctoAPI.YVoltage;
py from yocto_voltage import *

```

### Fonction globales

#### **yFindVoltage(func)**

Permet de retrouver un capteur de tension d'après un identifiant donné.

#### **yFirstVoltage()**

Commence l'énumération des capteurs de tension accessibles par la librairie.

### Méthodes des objets YVoltage

#### **voltage→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **voltage→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **voltage→get\_advertisedValue()**

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

#### **voltage→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **voltage→get\_currentValue()**

Retourne la valeur instantanée de la tension.

#### **voltage→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### **voltage→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### **voltage→get\_friendlyName()**

Retourne un identifiant global du capteur de tension au format NOM\_MODULE . NOM\_FONCTION.

#### **voltage→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **voltage→get\_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

#### **voltage→get\_hardwareId()**

### 3. Reference

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.
<b>voltage→get_highestValue()</b> Retourne la valeur maximale observée pour la tension.
<b>voltage→get_logFrequency()</b> Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>voltage→get_logicalName()</b> Retourne le nom logique du capteur de tension.
<b>voltage→get_lowestValue()</b> Retourne la valeur minimale observée pour la tension.
<b>voltage→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>voltage→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>voltage→get_recordedData(startTime, endTime)</b> Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>voltage→get_reportFrequency()</b> Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>voltage→get_resolution()</b> Retourne la résolution des valeurs mesurées.
<b>voltage→get_unit()</b> Retourne l'unité dans laquelle la tension est exprimée.
<b>voltage→get(userData)</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>voltage→isOnline()</b> Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.
<b>voltage→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.
<b>voltage→load(msValidity)</b> Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.
<b>voltage→loadCalibrationPoints(rawValues, refValues)</b> Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>voltage→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.
<b>voltage→nextVoltage()</b> Continue l'énumération des capteurs de tension commencée à l'aide de yFirstVoltage( ).
<b>voltage→registerTimedReportCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>voltage→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>voltage→set_highestValue(newval)</b> Modifie la mémoire de valeur maximale observée pour la tension.
<b>voltage→set_logFrequency(newval)</b>

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**voltage→set\_logicalName(newval)**

Modifie le nom logique du capteur de tension.

**voltage→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la tension.

**voltage→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**voltage→set\_resolution(newval)**

Modifie la résolution des valeurs mesurées.

**voltage→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**voltage→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YVoltage.FindVoltage() yFindVoltage()yFindVoltage( )

**YVoltage**

Permet de retrouver un capteur de tension d'après un identifiant donné.

**YVoltage\* yFindVoltage( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltage.isOnLine()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de tension sans ambiguïté

### Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

**YVoltage.FirstVoltage()****YVoltage****yFirstVoltage()yFirstVoltage( )**

Commence l'énumération des capteurs de tension accessibles par la librairie.

**YVoltage\* yFirstVoltage( )**

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

**Retourne :**

un pointeur sur un objet `YVoltage`, correspondant au premier capteur de tension accessible en ligne, ou null si il n'y a pas de capteurs de tension disponibles.

**voltage→calibrateFromPoints()voltage→  
calibrateFromPoints( )****YVoltage**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( vector<double> rawValues,  
                           vector<double> 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.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→describe()voltage→describe()****YVoltage**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**voltage→get\_advertisedValue()** **YVoltage**  
**voltage→advertisedValue()voltage→**  
**get\_advertisedValue( )**

---

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

string **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.

---

<b>voltage→get_currentRawValue()</b>	<b>YVoltage</b>
<b>voltage→currentRawValue()voltage→</b>	
<b>get_currentRawValue( )</b>	

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
double get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**voltage→get\_currentValue()** **YVoltage**  
**voltage→currentValue()voltage→**  
**get\_currentValue( )**

---

Retourne la valeur instantanée de la tension.

```
double get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur instantanée de la tension

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**voltage→getErrorMessage()**  
**voltage→errorMessage()voltage→**  
**getErrorMessage( )**

**YVoltage**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

**string getErrorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

**voltage→get\_errorType()**

**YVoltage**

**voltage→errorType()voltage→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

**voltage→get\_friendlyName()**  
**voltage→friendlyName()voltage→**  
**get\_friendlyName( )**

**YVoltage**

Retourne un identifiant global du capteur de tension au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du capteur de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de tension (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le capteur de tension en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

<b>voltage→get_functionDescriptor()</b>	<b>YVoltage</b>
<b>voltage→functionDescriptor()voltage→get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voltage→get\_functionId()****YVoltage****voltage→functionId()voltage→get\_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

```
string get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voltage→get\_hardwareId()**

**YVoltage**

**voltage→hardwareId()voltage→get\_hardwareId( )**

---

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.

string **get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de tension (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

<b>voltage→get_highestValue()</b>	<b>YVoltage</b>
<b>voltage→highestValue()voltage→</b>	
<b>get_highestValue()</b>	

---

Retourne la valeur maximale observée pour la tension.

```
double get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la tension

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**voltage→get\_logFrequency()** YVoltage  
**voltage→logFrequency()voltage→**  
**get\_logFrequency( )**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

string **get\_logFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**voltage→get\_logicalName()**  
**voltage→logicalName()voltage→**  
**get\_logicalName( )**

**YVoltage**

Retourne le nom logique du capteur de tension.

**string get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de tension. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**voltage→get\_lowestValue()** **YVoltage**  
**voltage→lowestValue()voltage→**  
**get\_lowestValue( )**

---

Retourne la valeur minimale observée pour la tension.

```
double get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la tension

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**voltage→get\_module()****YVoltage****voltage→module()voltage→get\_module( )**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

**voltage→get\_recordedData()**  
**voltage→recordedData()voltage→**  
**get\_recordedData( )**

**YVoltage**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**YDataSet get\_recordedData( s64 startTime, s64 endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

<b>voltage→get_reportFrequency()</b>	<b>YVoltage</b>
<b>voltage→reportFrequency()voltage→get_reportFrequency( )</b>	

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
string get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**voltage→get\_resolution()**

**YVoltage**

**voltage→resolution()voltage→get\_resolution( )**

---

Retourne la résolution des valeurs mesurées.

**double get\_resolution( )**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**voltage→get\_unit()****YVoltage****voltage→unit()voltage→get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
string get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**voltage→get(userData())**

**YVoltage**

**voltage→userData()voltage→get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* 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()****YVoltage**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de tension est joignable, false sinon

**voltage→load()voltage→load( )****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

**YRETCODE load( int 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.

---

<b>voltage</b> → <b>loadCalibrationPoints()</b> <b>voltage</b> → <b>loadCalibrationPoints( )</b>	<b>YVoltage</b>
-----------------------------------------------------------------------------------------------------	-----------------

---

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( vector<double>& rawValues,  
                           vector<double>& refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **voltage→nextVoltage()voltage→nextVoltage( )**

**YVoltage**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

`YVoltage * nextVoltage( )`

**Retourne :**

un pointeur sur un objet `YVoltage` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**voltage→registerTimedReportCallback()voltage→registerTimedReportCallback( )****YVoltage**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( YVoltageTimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**voltage→registerValueCallback()** **voltage→registerValueCallback( )** **YVoltage**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YVoltageValueCallback 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 appellés 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.

---

<b>voltage→set_highestValue()</b>	<b>YVoltage</b>
<b>voltage→setHighestValue()voltage→</b> <b>set_highestValue()</b>	

Modifie la mémoire de valeur maximale observée pour la tension.

```
int set_highestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la tension

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>voltage→set_logFrequency()</b>	<b>YVoltage</b>
<b>voltage→setLogFrequency()voltage→</b> <b>set_logFrequency( )</b>	

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**int set\_logFrequency( const string& newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_logicalName()**  
**voltage→setLogicalName()****voltage→**  
**set\_logicalName( )**

**YVoltage**

Modifie le nom logique du capteur de tension.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser **yCheckLogicalName( )** pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode **saveToFlash( )** du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de tension.

**Retourne :**

**YAPI\_SUCCESS** si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_lowestValue()** YVoltage  
**voltage→setLowestValue()** **voltage→set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour la tension.

```
int set_lowestValue( double newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la tension

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_reportFrequency()****YVoltage****voltage→setReportFrequency()voltage→  
set\_reportFrequency( )**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( const string& newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>voltage→set_resolution()</b>	<b>YVoltage</b>
<b>voltage→setResolution()voltage→set_resolution()</b>	

Modifie la résolution des valeurs mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set(userData())****YVoltage****voltage→setUserData()voltage→set(userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void set(userData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.42. Interface de la fonction Source de tension

La librairie de programmation Yoctopuce permet de commander la tension de sortie du module. Vous pouvez affecter une valeur fixe, ou faire des transitions de voltage.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_vsource.js'></script>
php	require_once('yocto_vsource.php');
cpp	#include "yocto_vsource.h"
m	#import "yocto_vsource.h"
pas	uses yocto_vsource;
vb	yocto_vsource.vb
cs	yocto_vsource.cs
java	import com.yoctopuce.YoctoAPI.YVSource;
py	from yocto_vsource import *

<b>Fonction globales</b>	
<b>yFindVSource(func)</b>	Permet de retrouver une source de tension d'après un identifiant donné.
<b>yFirstVSource()</b>	Commence l'énumération des sources de tension accessibles par la librairie.
<b>Méthodes des objets YVSource</b>	
<b>vsource→describe()</b>	Retourne un court texte décrivant la fonction au format TYPE ( NAME ) = SERIAL . FUNCTIONID.
<b>vsource→get_advertisedValue()</b>	Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
<b>vsource→get_errorMessage()</b>	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>vsource→get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>vsource→get_extPowerFailure()</b>	Rend TRUE si le voltage de l'alimentation externe est trop bas.
<b>vsource→get_failure()</b>	Indique si le module est en condition d'erreur.
<b>vsource→get_friendlyName()</b>	Retourne un identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.
<b>vsource→get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>vsource→get_functionId()</b>	Retourne l'identifiant matériel de la fonction, sans référence au module.
<b>vsource→get_hardwareId()</b>	Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.
<b>vsource→get_logicalName()</b>	Retourne le nom logique de la source de tension.
<b>vsource→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>vsource→get_module_async(callback, context)</b>	

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**vsouce→get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

**vsouce→get\_overHeat()**

Rend TRUE si le module est en surchauffe.

**vsouce→get\_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

**vsouce→get\_regulationFailure()**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

**vsouce→get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

**vsouce→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**vsouce→get\_voltage()**

Retourne la valeur de la commande de tension de sortie en mV

**vsouce→isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

**vsouce→isOnline\_async(callback, context)**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

**vsouce→load(msValidity)**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**vsouce→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**vsouce→nextVSource()**

Continue l'énumération des sources de tension commencée à l'aide de yFirstVSource( ).

**vsouce→pulse(voltage, ms\_duration)**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

**vsouce→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**vsouce→set\_logicalName(newval)**

Modifie le nom logique de la source de tension.

**vsouce→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).

**vsouce→set\_voltage(newval)**

Règle la tension de sortie du module (en millivolts).

**vsouce→voltageMove(target, ms\_duration)**

Déclenche une variation constante de la sortie vers une valeur donnée.

**vsouce→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**yFindVSource() —****YVSource****YVSource.FindVSource()yFindVSource( )**

Permet de retrouver une source de tension d'après un identifiant donné.

**YVSource\* yFindVSource( const string& func)**

**yFindVSource() — YVSource.FindVSource()yFindVSource( )**

Permet de retrouver une source de tension d'après un identifiant donné.

<b>js</b>	<b>function yFindVSource( func)</b>
<b>php</b>	<b>function yFindVSource( \$func)</b>
<b>cpp</b>	<b>YVSource* yFindVSource( const string&amp; func)</b>
<b>m</b>	<b>YVSource* yFindVSource( NSString* func)</b>
<b>pas</b>	<b>function yFindVSource( func: string): TYVSource</b>
<b>vb</b>	<b>function yFindVSource( ByVal func As String) As YVSource</b>
<b>cs</b>	<b>YVSource FindVSource( string func)</b>
<b>java</b>	<b>YVSource FindVSource( String func)</b>
<b>py</b>	<b>def FindVSource( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

**Retourne :**

un objet de classe `YVSource` qui permet ensuite de contrôler la source de tension.

**yFirstVSource() —****YVSource****YVSource.FirstVSource()yFirstVSource( )**

Commence l'énumération des sources de tension accessibles par la librairie.

**YVSource\* yFirstVSource( )**

**yFirstVSource() — YVSource.FirstVSource()yFirstVSource( )**

Commence l'énumération des sources de tension accessibles par la librairie.

js	function <b>yFirstVSource( )</b>
php	function <b>yFirstVSource( )</b>
cpp	<b>YVSource*</b> <b>yFirstVSource( )</b>
m	<b>YVSource*</b> <b>yFirstVSource( )</b>
pas	function <b>yFirstVSource( )</b> : TYVSource
vb	function <b>yFirstVSource( )</b> As YVSource
cs	<b>YVSource FirstVSource( )</b>
java	<b>YVSource FirstVSource( )</b>
py	def <b>FirstVSource( )</b>

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.

**vsources->describe()****YVSource**

Retourne un court texte décrivant la fonction au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**string describe( )**

**vsources->describe()**

Retourne un court texte décrivant la fonction au format TYPE ( NAME )=SERIAL . FUNCTIONID.

<b>js</b>	function <b>describe( )</b>
<b>php</b>	function <b>describe( )</b>
<b>cpp</b>	string <b>describe( )</b>
<b>m</b>	<b>-(NSString*) describe</b>
<b>pas</b>	function <b>describe( )</b> : string
<b>vb</b>	function <b>describe( ) As String</b>
<b>cs</b>	string <b>describe( )</b>
<b>java</b>	String <b>describe( )</b>

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant la fonction (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**vsource→get\_advertisedValue()**  
**vsource→advertisedValue()vsource→get\_advertisedValue( )**

**YVSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

string **get\_advertisedValue( )**

**vsource→get\_advertisedValue()**

**vsource→advertisedValue()vsource→get\_advertisedValue( )**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

**js** function **get\_advertisedValue( )**  
**php** function **get\_advertisedValue( )**  
**cpp** string **get\_advertisedValue( )**  
**m** -(NSString\*) advertisedValue  
**pas** function **get\_advertisedValue( )**: string  
**vb** function **get\_advertisedValue( )** As String  
**cs** string **get\_advertisedValue( )**  
**java** String **get\_advertisedValue( )**  
**py** def **get\_advertisedValue( )**  
**cmd** YVSource **target get\_advertisedValue**

**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**.

**vsouce→get\_errorMessage()**  
**vsouce→errorMessage()vsouce→get\_errorMessage( )**

**YVSource**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

string **get\_errorMessage( )**

**vsouce→get\_errorMessage()**  
**vsouce→errorMessage()vsouce→get\_errorMessage( )**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

js   **function get\_errorMessage( )**  
php   **function get\_errorMessage( )**  
cpp   **string get\_errorMessage( )**  
m    **-(NSString\*) errorMessage**  
pas   **function get\_errorMessage( ): string**  
vb    **function get\_errorMessage( ) As String**  
cs    **string get\_errorMessage( )**  
java   **String get\_errorMessage( )**  
py    **def get\_errorMessage( )**

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

**vsouce→get\_errorType()****YVSource****vsouce→errorType()vsouce→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

YRETCODE **get\_errorType( )**

**vsouce→get\_errorType()****vsouce→errorType()vsouce→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

js	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	YRETCODE <b>get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	YRETCODE <b>get_errorType( )</b>
java	int <b>get_errorType( )</b>
py	def <b>get_errorType( )</b>

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

---

<b>vsources-&gt;get_extPowerFailure()</b>	<b>YVSource</b>
<b>vsources-&gt;extPowerFailure()</b>	<b>YVSource-&gt;</b>
<b>get_extPowerFailure( )</b>	

---

Rend TRUE si le voltage de l'alimentation externe est trop bas.

[Y\\_EXTPOWERFAILURE\\_enum get\\_extPowerFailure\( \)](#)

---

<b>vsources-&gt;get_extPowerFailure()</b>	
<b>vsources-&gt;extPowerFailure()</b>	<b>YVSource-&gt;get_extPowerFailure( )</b>

---

Rend TRUE si le voltage de l'alimentation externe est trop bas.

```
js function get_extPowerFailure( )
php function get_extPowerFailure( )
cpp Y_EXTPOWERFAILURE_enum get_extPowerFailure( )
m -(Y_EXTPOWERFAILURE_enum) extPowerFailure
pas function get_extPowerFailure( ): Integer
vb function get_extPowerFailure( ) As Integer
cs int get_extPowerFailure( )
java int get_extPowerFailure( )
py def get_extPowerFailure( )
cmd YVSource target get_extPowerFailure
```

**Retourne :**

soit Y\_EXTPOWERFAILURE\_FALSE, soit Y\_EXTPOWERFAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_EXTPOWERFAILURE\_INVALID.

**vsources->get\_failure()****YVSource****vsources->failure()>vsources->get\_failure()**

Indique si le module est en condition d'erreur.

```
Y_FAILURE_enum get_failure( )
```

**vsources->get\_failure()****vsources->failure()>vsources->get\_failure()**

Indique si le module est en condition d'erreur.

```

js   function get_failure( )
php  function get_failure( )
cpp  Y_FAILURE_enum get_failure( )
m    -(Y_FAILURE_enum) failure
pas   function get_failure( ): Integer
vb    function get_failure( ) As Integer
cs    int get_failure( )
java  int get_failure( )
py    def get_failure( )
cmd   YVSource target get_failure

```

Il possible de savoir de quelle erreur il s'agit en testant get\_overheat, get\_overcurrent etc... Lorsqu'un condition d'erreur est rencontrée, la tension de sortie est mise à zéro et ne peut pas être changée tant la fonction reset() n'aura pas appellée.

**Retourne :**

soit Y\_FAILURE\_FALSE, soit Y\_FAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_FAILURE\_INVALID.

**vsource→get\_friendlyName()**  
**vsource→friendlyName()vsource→get\_friendlyName( )**

**YVSource**

Retourne un identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

virtual string **get\_friendlyName( )**

**vsource→get\_friendlyName()**  
**vsource→friendlyName()vsource→get\_friendlyName( )**

Retourne un identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

**js** function **get\_friendlyName( )**  
**php** function **get\_friendlyName( )**  
**cpp** virtual string **get\_friendlyName( )**  
**m** -(NSString\*) friendlyName  
**cs** override string **get\_friendlyName( )**  
**java** String **get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et de la fonction si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la fonction (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant la fonction en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**vsource→get\_functionDescriptor()**  
**vsource→functionDescriptor()vsource→get\_vsourceDescriptor( )**

**YVSource**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

[YFUN\\_DESCR get\\_functionDescriptor\( \)](#)

**vsource→get\_functionDescriptor()**  
**vsource→functionDescriptor()vsource→get\_vsourceDescriptor( )**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

<a href="#">js</a>	function <b>get_functionDescriptor( )</b>
<a href="#">php</a>	function <b>get_functionDescriptor( )</b>
<a href="#">cpp</a>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<a href="#">m</a>	-(YFUN_DESCR) functionDescriptor
<a href="#">pas</a>	function <b>get_functionDescriptor( )</b> : YFUN_DESCR
<a href="#">vb</a>	function <b>get_functionDescriptor( )</b> As YFUN_DESCR
<a href="#">cs</a>	YFUN_DESCR <b>get_functionDescriptor( )</b>
<a href="#">java</a>	String <b>get_functionDescriptor( )</b>
<a href="#">py</a>	def <b>get_functionDescriptor( )</b>

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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**vsouce→get\_functionId()****YVSource****vsouce→functionId()vsouce→get\_vsourceId( )**

Retourne l'identifiant matériel de la fonction, sans référence au module.

string **get\_functionId( )**

**vsouce→get\_functionId()****vsouce→functionId()vsouce→get\_vsourceId( )**

Retourne l'identifiant matériel de la fonction, sans référence au module.

js      function **get\_functionId( )**  
php     function **get\_functionId( )**  
cpp     string **get\_functionId( )**  
m       -(NSString\*) **functionId**  
vb      function **get\_functionId( ) As String**  
cs      string **get\_functionId( )**  
java    String **get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**vsources→get.hardwareId()**  
**vsources→hardwareId()** vsources→  
**get.hardwareId( )**

**YVSource**

Retourne l'identifiant matériel unique de la fonction au format SERIAL.FUNCTIONID.

string **get.hardwareId( )**

**vsources→get.hardwareId()**  
**vsources→hardwareId()** vsources→**get.hardwareId( )**

Retourne l'identifiant matériel unique de la fonction au format SERIAL.FUNCTIONID.

**js** function **get.hardwareId( )**  
**php** function **get.hardwareId( )**  
**cpp** string **get.hardwareId( )**  
**m** -(NSString\*) hardwareId  
**vb** function **get.hardwareId( ) As String**  
**cs** string **get.hardwareId( )**  
**java** String **get.hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

**vsources->get\_logicalName()** YVSource  
**vsources->logicalName()** vsources->  
get\_logicalName( )

---

Retourne le nom logique de la source de tension.

string get\_logicalName( )

**vsources->get\_logicalName()**  
**vsources->logicalName()** vsources->get\_logicalName( )

---

Retourne le nom logique de la source de tension.

js function get\_logicalName( )  
php function get\_logicalName( )  
cpp string get\_logicalName( )  
m -(NSString\*) logicalName  
pas function get\_logicalName( ): string  
vb function get\_logicalName( ) As String  
cs string get\_logicalName( )  
java String get\_logicalName( )  
py def get\_logicalName( )  
cmd YVSource target get\_logicalName

**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()****YVSource****vsource→module()vsource→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule * get_module( )`

**vsource→get\_module()****vsource→module()vsource→get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`js function get_module( )`

`php function get_module( )`

`cpp YModule * get_module( )`

`m -(YModule*) module`

`pas function get_module( ): TYModule`

`vb function get_module( ) As YModule`

`cs YModule get_module( )`

`java YModule get_module( )`

`py def get_module( )`

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

<b>vsOURCE→get_overCurrent()</b>	<b>YVSource</b>
<b>vsOURCE→overCurrent()vsOURCE→get_overCurrent( )</b>	

---

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

[Y\\_OVERCURRENT\\_enum get\\_overCurrent\( \)](#)

---

<b>vsOURCE→get_overCurrent()</b>
<b>vsOURCE→overCurrent()vsOURCE→get_overCurrent( )</b>

---

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
js   function get_overCurrent( )
php  function get_overCurrent( )
cpp  Y_OVERCURRENT_enum get_overCurrent( )
m    -(Y_OVERCURRENT_enum) overCurrent
pas   function get_overCurrent( ): Integer
vb    function get_overCurrent( ) As Integer
cs    int get_overCurrent( )
java  int get_overCurrent( )
py    def get_overCurrent( )
cmd   YVSource target get_overCurrent
```

**Retourne :**

soit Y\_OVERCURRENT\_FALSE, soit Y\_OVERCURRENT\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERCURRENT\_INVALID.

**vsource→get\_overHeat()****YVSource****vsource→overHeat()vsource→get\_overHeat( )**

Rend TRUE si le module est en surchauffe.

[Y\\_OVERHEAT\\_enum get\\_overHeat\( \)](#)

**vsource→get\_overHeat()****vsource→overHeat()vsource→get\_overHeat( )**

Rend TRUE si le module est en surchauffe.

```
js function get_overHeat( )
php function get_overHeat( )
cpp Y_OVERHEAT_enum get_overHeat( )
m -(Y_OVERHEAT_enum) overHeat
pas function get_overHeat( ): Integer
vb function get_overHeat( ) As Integer
cs int get_overHeat( )
java int get_overHeat( )
py def get_overHeat( )
cmd YVSource target get_overHeat
```

**Retourne :**

soit Y\_OVERHEAT\_FALSE, soit Y\_OVERHEAT\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERHEAT\_INVALID.

**vsOURCE→get\_overLoad()****YVSource****vsOURCE→overLoad()vsOURCE→get\_overLoad( )**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

**Y\_OVERLOAD\_enum get\_overLoad( )****vsOURCE→get\_overLoad()****vsOURCE→overLoad()vsOURCE→get\_overLoad( )**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

js	function get_overLoad( )
php	function get_overLoad( )
cpp	Y_OVERLOAD_enum get_overLoad( )
m	-(Y_OVERLOAD_enum) overLoad
pas	function get_overLoad( ): Integer
vb	function get_overLoad( ) As Integer
cs	int get_overLoad( )
java	int get_overLoad( )
py	def get_overLoad( )
cmd	YVSource target get_overLoad

**Retourne :**

soit Y\_OVERLOAD\_FALSE, soit Y\_OVERLOAD\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERLOAD\_INVALID.

**vsource→get\_regulationFailure()**  
**vsource→regulationFailure()vsource→get\_regulationFailure( )**

**YVSource**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

**Y\_REGULATIONFAILURE\_enum get\_regulationFailure( )**

**vsource→get\_regulationFailure()**  
**vsource→regulationFailure()vsource→get\_regulationFailure( )**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

<b>js</b>	<b>function get_regulationFailure( )</b>
<b>php</b>	<b>function get_regulationFailure( )</b>
<b>cpp</b>	<b>Y_REGULATIONFAILURE_enum get_regulationFailure( )</b>
<b>m</b>	<b>-(Y_REGULATIONFAILURE_enum) regulationFailure</b>
<b>pas</b>	<b>function get_regulationFailure( ): Integer</b>
<b>vb</b>	<b>function get_regulationFailure( ) As Integer</b>
<b>cs</b>	<b>int get_regulationFailure( )</b>
<b>java</b>	<b>int get_regulationFailure( )</b>
<b>py</b>	<b>def get_regulationFailure( )</b>
<b>cmd</b>	<b>YVSource target get_regulationFailure</b>

**Retourne :**

soit Y\_REGULATIONFAILURE\_FALSE, soit Y\_REGULATIONFAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_REGULATIONFAILURE\_INVALID.

**vsOURCE→get\_unit()**  
**vsOURCE→unit()vsOURCE→get\_unit( )**

**YVSource**

Retourne l'unité dans laquelle la tension est exprimée.

string **get\_unit( )**

**vsOURCE→get\_unit()**  
**vsOURCE→unit()vsOURCE→get\_unit( )**

Retourne l'unité dans laquelle la tension est exprimée.

js      function **get\_unit( )**  
php    function **get\_unit( )**  
cpp    string **get\_unit( )**  
m      -(NSString\*) unit  
pas    function **get\_unit( )**: string  
vb     function **get\_unit( )** As String  
cs     string **get\_unit( )**  
java   String **get\_unit( )**  
py     def **get\_unit( )**  
cmd    YVSource target **get\_unit**

**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**.

**vsouce→get(userData)****YVSource****vsouce→userData()vsouce→get(userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * get(userData)
```

**vsouce→get(userData)****vsouce→userData()vsouce→get(userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

js	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): Tobject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

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.

---

**vsources->get\_voltage()** **YVSource**  
**vsources->voltage()>vsources->get\_voltage( )**

---

Retourne la valeur de la commande de tension de sortie en mV

```
int get_voltage( )
```

**vsources->get\_voltage()**  
**vsources->voltage()>vsources->get\_voltage( )**

---

Retourne la valeur de la commande de tension de sortie en mV

```
js    function get_voltage( )
php   function get_voltage( )
cpp   int get_voltage( )
m     -(int) voltage
pas   function get_voltage( ): LongInt
vb    function get_voltage( ) As Integer
cs    int get_voltage( )
java  int get_voltage( )
py    def get_voltage( )
```

**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()****YVSource**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

bool **isOnline( )**

**vsource→isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	def <b>isOnline( )</b>

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

**vsouce→load()vsouce→load( )****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

**vsouce→load()vsouce→load( )**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

js	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( msValidity)</b>

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.

**vsource→nextVSource()vsource→nextVSource( )****YVSource**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

`YVSource * nextVSource( )`

**vsource→nextVSource()vsource→nextVSource( )**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

<code>js</code>	<code>function nextVSource( )</code>
<code>php</code>	<code>function nextVSource( )</code>
<code>cpp</code>	<code>YVSource * nextVSource( )</code>
<code>m</code>	<code>-(YVSource*) nextVSource</code>
<code>pas</code>	<code>function nextVSource( ): TYVSource</code>
<code>vb</code>	<code>function nextVSource( ) As YVSource</code>
<code>cs</code>	<code>YVSource nextVSource( )</code>
<code>java</code>	<code>YVSource nextVSource( )</code>
<code>py</code>	<code>def nextVSource( )</code>

**Retourne :**

un pointeur sur un objet `YVSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**vsOURCE→pulse()****YVSource**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
int pulse( int voltage, int ms_duration)
```

**vsOURCE→pulse()**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

js	function pulse( voltage, ms_duration)
php	function pulse( \$voltage, \$ms_duration)
cpp	int pulse( int voltage, int ms_duration)
m	-{int) pulse : (int) voltage : (int) ms_duration
pas	function pulse( voltage: integer, ms_duration: integer): integer
vb	function pulse( ByVal voltage As Integer, ByVal ms_duration As Integer) As Integer
cs	int pulse( int voltage, int ms_duration)
java	int pulse( int voltage, int ms_duration)
py	def pulse( voltage, ms_duration)
cmd	YVSource target pulse voltage ms_duration

**Paramètres :**

**voltage** tension demandée, en millivolts

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsource→registerValueCallback()vsource→registerValueCallback( )****YVSource**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
void registerValueCallback( YDisplayUpdateCallback callback)
```

**vsource→registerValueCallback()vsource→registerValueCallback( )**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

<code>js</code>	<code>function registerValueCallback( <b>callback</b>)</code>
<code>php</code>	<code>function registerValueCallback( \$callback)</code>
<code>cpp</code>	<code>void registerValueCallback( YDisplayUpdateCallback <b>callback</b>)</code>
<code>pas</code>	<code>procedure registerValueCallback( <b>callback</b>: TGenericUpdateCallback)</code>
<code>vb</code>	<code>procedure registerValueCallback( ByVal <b>callback</b> As GenericUpdateCallback)</code>
<code>cs</code>	<code>void registerValueCallback( UpdateCallback <b>callback</b>)</code>
<code>java</code>	<code>void registerValueCallback( UpdateCallback <b>callback</b>)</code>
<code>py</code>	<code>def registerValueCallback( <b>callback</b>)</code>
<code>m</code>	<code>-(void) registerValueCallback : (YFunctionUpdateCallback) <b>callback</b></code>

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 appellés 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.

**vsouce→set\_logicalName()**  
**vsouce→setLogicalName()** vsouce→  
**set\_logicalName( )**

YVSource

Modifie le nom logique de la source de tension.

int **set\_logicalName( const string& newval)**

**vsouce→set\_logicalName()**  
**vsouce→setLogicalName()** vsouce→  
**set\_logicalName( )**

Modifie le nom logique de la source de tension.

js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	- (int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval): string</b> : integer
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	YVSource target <b>set_logicalName newval</b>

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.

**vsource→set(userData())****YVSource****vsource→setUserData()vsource→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).

```
void set(userData( void* data)
```

**vsource→set(userData())****vsource→setUserData()vsource→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).

<b>js</b>	function set(userData( data)
<b>php</b>	function set(userData( \$data)
<b>cpp</b>	void set(userData( void* data)
<b>m</b>	-(void) setUserData : (void*) data
<b>pas</b>	procedure set(userData( data: Tobject)
<b>vb</b>	procedure set(userData( ByVal data As Object)
<b>cs</b>	void set(userData( object data)
<b>java</b>	void set(userData( Object data)
<b>py</b>	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

**vsouce→set\_voltage()** **YVSource**  
**vsouce→setVoltage()vsouce→set\_voltage( )**

Règle la tension de sortie du module (en millivolts).

int **set\_voltage( int newval)**

**vsouce→set\_voltage()**  
**vsouce→setVoltage()vsouce→set\_voltage( )**

Règle la tension de sortie du module (en millivolts).

```
js   function set_voltage( newval)
php  function set_voltage( $newval)
cpp  int set_voltage( int newval)
m    -(int) setVoltage : (int) newval
pas   function set_voltage( newval: LongInt): integer
vb    function set_voltage( ByVal newval As Integer) As Integer
cs    int set_voltage( int newval)
java  int set_voltage( int newval)
py    def set_voltage( newval)
cmd   YVSource target set_voltage newval
```

#### Paramètres :

**newval** un entier

#### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsource→voltageMove()****vsource→voltageMove( )****YVSource**

Déclenche une variation constante de la sortie vers une valeur donnée.

```
int voltageMove( int target, int ms_duration)
```

**vsource→voltageMove()****vsource→voltageMove( )**

Déclenche une variation constante de la sortie vers une valeur donnée.

<code>js</code>	<code>function voltageMove( target, ms_duration)</code>
<code>php</code>	<code>function voltageMove( \$target, \$ms_duration)</code>
<code>cpp</code>	<code>int voltageMove( int target, int ms_duration)</code>
<code>m</code>	<code>-(int) voltageMove : (int) target : (int) ms_duration</code>
<code>pas</code>	<code>function voltageMove( target: integer, ms_duration: integer): integer</code>
<code>vb</code>	<code>function voltageMove( ByVal target As Integer,</code> <code>                  ByVal ms_duration As Integer) As Integer</code>
<code>cs</code>	<code>int voltageMove( int target, int ms_duration)</code>
<code>java</code>	<code>int voltageMove( int target, int ms_duration)</code>
<code>py</code>	<code>def voltageMove( target, ms_duration)</code>
<code>cmd</code>	<code>YVSource target voltageMove target ms_duration</code>

**Paramètres :**

**target** nouvelle valeur de sortie à la fin de la transition, en millivolts.

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.43. Interface de la fonction WakeUpMonitor

La fonction WakeUpMonitor prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_wakeupmonitor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YWakeUpMonitor = yoctolib.YWakeUpMonitor;
php require_once('yocto_wakeupmonitor.php');
cpp #include "yocto_wakeupmonitor.h"
m #import "yocto_wakeupmonitor.h"
pas uses yocto_wakeupmonitor;
vb yocto_wakeupmonitor.vb
cs yocto_wakeupmonitor.cs
java import com.yoctopuce.YoctoAPI.YWakeUpMonitor;
py from yocto_wakeupmonitor import *

```

### Fonction globales

#### yFindWakeUpMonitor(func)

Permet de retrouver un moniteur d'après un identifiant donné.

#### yFirstWakeUpMonitor()

Commence l'énumération des Moniteurs accessibles par la librairie.

### Méthodes des objets YWakeUpMonitor

#### wakeupmonitor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format TYPE (NAME )=SERIAL . FUNCTIONID.

#### wakeupmonitor→get\_advertisedValue()

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

#### wakeupmonitor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

#### wakeupmonitor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

#### wakeupmonitor→get\_friendlyName()

Retourne un identifiant global du moniteur au format NOM\_MODULE . NOM\_FONCTION.

#### wakeupmonitor→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### wakeupmonitor→get\_functionId()

Retourne l'identifiant matériel du moniteur, sans référence au module.

#### wakeupmonitor→get\_hardwareId()

Retourne l'identifiant matériel unique du moniteur au format SERIAL . FUNCTIONID.

#### wakeupmonitor→get\_logicalName()

Retourne le nom logique du moniteur.

#### wakeupmonitor→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupmonitor→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

<b>wakeupmonitor→get_nextWakeUp()</b>	Retourne la prochaine date/heure de réveil agendée (format UNIX)
<b>wakeupmonitor→get_powerDuration()</b>	Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.
<b>wakeupmonitor→get_sleepCountdown()</b>	Retourne le temps avant le prochain sommeil.
<b>wakeupmonitor→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>wakeupmonitor→get_wakeUpReason()</b>	Renvoie la raison du dernier réveil.
<b>wakeupmonitor→get_wakeUpState()</b>	Revoie l'état actuel du moniteur
<b>wakeupmonitor→isOnline()</b>	Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.
<b>wakeupmonitor→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.
<b>wakeupmonitor→load(msValidity)</b>	Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.
<b>wakeupmonitor→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.
<b>wakeupmonitor→nextWakeUpMonitor()</b>	Continue l'énumération des Moniteurs commencée à l'aide de <code>yFirstWakeUpMonitor()</code> .
<b>wakeupmonitor→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>wakeupmonitor→resetSleepCountDown()</b>	Réinitialise le compteur de mise en sommeil.
<b>wakeupmonitor→set_logicalName(newval)</b>	Modifie le nom logique du moniteur.
<b>wakeupmonitor→set_nextWakeUp(newval)</b>	Modifie les jours de la semaine où un réveil doit avoir lieu.
<b>wakeupmonitor→set_powerDuration(newval)</b>	Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.
<b>wakeupmonitor→set_sleepCountdown(newval)</b>	Modifie le temps avant le prochain sommeil .
<b>wakeupmonitor→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>wakeupmonitor→sleep(secBeforeSleep)</b>	Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.
<b>wakeupmonitor→sleepFor(secUntilWakeUp, secBeforeSleep)</b>	Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.
<b>wakeupmonitor→sleepUntil(wakeUpTime, secBeforeSleep)</b>	Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.
<b>wakeupmonitor→wait_async(callback, context)</b>	

### **3. Reference**

---

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

#### **wakeupmonitor→wakeUp()**

Force un réveil.

## YWakeUpMonitor.FindWakeUpMonitor() yFindWakeUpMonitor()yFindWakeUpMonitor( )

## YWakeUpMonitor

Permet de retrouver un moniteur d'après un identifiant donné.

**YWakeUpMonitor\* yFindWakeUpMonitor( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.isOnline()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le moniteur sans ambiguïté

### Retourne :

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

## **YWakeUpMonitor.FirstWakeUpMonitor()**

## **YWakeUpMonitor**

### **yFirstWakeUpMonitor()yFirstWakeUpMonitor( )**

Commence l'énumération des Moniteurs accessibles par la librairie.

**YWakeUpMonitor\* yFirstWakeUpMonitor( )**

Utiliser la fonction `YWakeUpMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

**Retourne :**

un pointeur sur un objet `YWakeUpMonitor`, correspondant au premier moniteur accessible en ligne, ou `null` si il n'y a pas de Moniteurs disponibles.

**wakeupmonitor→describe()**  
**wakeupmonitor→  
describe( )****YWakeUpMonitor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format TYPE ( NAME ) =SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le moniteur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

wakeupmonitor→get\_advertisedValue()  
wakeupmonitor→advertisedValue(wakeupmonitor  
→get\_advertisedValue())

YWakeUpMonitor

---

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

string **get\_advertisedValue()**

**Retourne :**

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

wakeupmonitor→getErrorMessage()

YWakeUpMonitor

wakeupmonitor→errorMessage()wakeupmonitor→  
getErrorMessage( )

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

string getErrorMessage( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

wakeupmonitor→get\_errorType()

YWakeUpMonitor

wakeupmonitor→errorType()wakeupmonitor→  
get\_errorType( )

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

wakeupmonitor→get\_friendlyName()

YWakeUpMonitor

wakeupmonitor→friendlyName()wakeupmonitor→  
get\_friendlyName( )

Retourne un identifiant global du moniteur au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du moniteur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du moniteur (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le moniteur en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

wakeupmonitor→get\_functionDescriptor()

YWakeUpMonitor

wakeupmonitor→functionDescriptor()

wakeupmonitor→get\_functionDescriptor( )

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**wakeupmonitor→get\_functionId()****YWakeUpMonitor****wakeupmonitor→functionId()wakeupmonitor→  
get\_functionId( )**

---

Retourne l'identifiant matériel du moniteur, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le moniteur (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

wakeupmonitor→get\_hardwareId()

YWakeUpMonitor

wakeupmonitor→hardwareId()wakeupmonitor→  
get\_hardwareId( )

---

Retourne l'identifiant matériel unique du moniteur au format SERIAL.FUNCTIONID.

string **get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moniteur (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le moniteur (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

wakeupmonitor→get\_logicalName()

YWakeUpMonitor

wakeupmonitor→logicalName()wakeupmonitor→  
get\_logicalName( )

---

Retourne le nom logique du moniteur.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du moniteur. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

wakeupmonitor→get\_module()

YWakeUpMonitor

wakeupmonitor→module()wakeupmonitor→  
get\_module()

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

wakeupmonitor→get\_nextWakeUp()

YWakeUpMonitor

wakeupmonitor→nextWakeUp()wakeupmonitor→  
get\_nextWakeUp( )

---

Retourne la prochaine date/heure de réveil agendée (format UNIX)

s64 get\_nextWakeUp( )

**Retourne :**

un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTWAKEUP\_INVALID.

wakeupmonitor→get\_powerDuration()

YWakeUpMonitor

wakeupmonitor→powerDuration()wakeupmonitor→  
get\_powerDuration( )

---

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

**int get\_powerDuration( )**

**Retourne :**

un entier représentant le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement

En cas d'erreur, déclenche une exception ou retourne Y\_POWERDURATION\_INVALID.

wakeupmonitor→get\_sleepCountdown()

YWakeUpMonitor

wakeupmonitor→sleepCountdown()wakeupmonitor

→get\_sleepCountdown( )

---

Retourne le temps avant le prochain sommeil.

```
int get_sleepCountdown( )
```

**Retourne :**

un entier représentant le temps avant le prochain sommeil

En cas d'erreur, déclenche une exception ou retourne Y\_SLEEP\_COUNTDOWN\_INVALID.

wakeupmonitor→get(userData)

YWakeUpMonitor

wakeupmonitor→userData(wakeupmonitor→

get(userData))

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**void \* get(userData)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

wakeupmonitor→get\_wakeUpReason()

YWakeUpMonitor

wakeupmonitor→wakeUpReason()wakeupmonitor→  
get\_wakeUpReason( )

Renvoie la raison du dernier réveil.

[Y\\_WAKEUPREASON\\_enum get\\_wakeUpReason\( \)](#)

**Retourne :**

une valeur parmi Y\_WAKEUPREASON\_USBPOWER, Y\_WAKEUPREASON\_EXTPOWER,  
Y\_WAKEUPREASON\_ENDOFSLEEP, Y\_WAKEUPREASON\_EXTSIG1,  
Y\_WAKEUPREASON\_EXTSIG2, Y\_WAKEUPREASON\_EXTSIG3,  
Y\_WAKEUPREASON\_EXTSIG4, Y\_WAKEUPREASON\_SCHEDULE1,  
Y\_WAKEUPREASON\_SCHEDULE2, Y\_WAKEUPREASON\_SCHEDULE3,  
Y\_WAKEUPREASON\_SCHEDULE4, Y\_WAKEUPREASON\_SCHEDULE5 et  
Y\_WAKEUPREASON\_SCHEDULE6

En cas d'erreur, déclenche une exception ou retourne Y\_WAKEUPREASON\_INVALID.

wakeupmonitor→get\_wakeUpState()

YWakeUpMonitor

wakeupmonitor→wakeUpState()wakeupmonitor→  
get\_wakeUpState( )

Revoie l'état actuel du moniteur

Y\_WAKEUPSTATE\_enum get\_wakeUpState( )

**Retourne :**

soit Y\_WAKEUPSTATE\_SLEEPING, soit Y\_WAKEUPSTATE\_AWAKE

En cas d'erreur, déclenche une exception ou retourne Y\_WAKEUPSTATE\_INVALID.

wakeupmonitor→isOnline()wakeupmonitor→  
isOnline( )

YWakeUpMonitor

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache du moniteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le moniteur est joignable, false sinon

**wakeupmonitor→load()****YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→nextWakeUpMonitor()**  
**wakeupmonitor→nextWakeUpMonitor( )**

**YWakeUpMonitor**

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor( )`.

`YWakeUpMonitor * nextWakeUpMonitor( )`

**Retourne :**

un pointeur sur un objet `YWakeUpMonitor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wakeupmonitor→registerValueCallback()****YWakeUpMonitor****wakeupmonitor→registerValueCallback( )**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YWakeUpMonitorValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

**wakeupmonitor→resetSleepCountDown()****YWakeUpMonitor****wakeupmonitor→resetSleepCountDown( )**

---

Réinitialise le compteur de mise en sommeil.**int resetSleepCountDown( )****Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_logicalName()	YWakeUpMonitor
wakeupmonitor→setLogicalName()wakeupmonitor →set_logicalName( )	

Modifie le nom logique du moniteur.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du moniteur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set\_nextWakeUp()**

**YWakeUpMonitor**

wakeupmonitor→**setNextWakeUp()**wakeupmonitor

→**set\_nextWakeUp( )**

---

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
int set_nextWakeUp( s64 newval)
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→set\_powerDuration()**  
**wakeupmonitor→setPowerDuration()**  
**wakeupmonitor→set\_powerDuration( )**

---

**YWakeUpMonitor**

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

**int set\_powerDuration( int newval)**

**Paramètres :**

**newval** un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set\_sleepCountdown()**  
wakeupmonitor→**setSleepCountdown()**  
wakeupmonitor→**set\_sleepCountdown( )**

**YWakeUpMonitor**

Modifie le temps avant le prochain sommeil .

int **set\_sleepCountdown( int newval)**

**Paramètres :**

**newval** un entier représentant le temps avant le prochain sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→set(userData())** YWakeUpMonitor  
**wakeupmonitor→setUserData()wakeupmonitor→**  
**set(userData())**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**wakeupmonitor→sleep()wakeupmonitor→sleep()****YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

**int sleep( int secBeforeSleep)**

**Paramètres :**

**secBeforeSleep** nombre de seconde avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→sleepFor()wakeupmonitor→  
sleepFor( )

YWakeUpMonitor

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

**int sleepFor( int secUntilWakeUp, int secBeforeSleep)**

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

**Paramètres :**

**secUntilWakeUp** durée de la mise en sommeil, en secondes

**secBeforeSleep** nombre de secondes avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→sleepUntil()wakeupmonitor→  
sleepUntil()****YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
int sleepUntil( int wakeUpTime, int secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

**Paramètres :**

**wakeUpTime** date/heure du réveil (format UNIX)

**secBeforeSleep** nombre de secondes avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor → **wakeUp()** wakeupmonitor →  
**wakeUp( )**

---

**YWakeUpMonitor**

Force un réveil.

```
int wakeUp( )
```

## 3.44. Interface de la fonction WakeUpSchedule

La fonction WakeUpSchedule implémente une condition de réveil. Le réveil est spécifiée par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_wakeupschedule.js'></script>
nodejs var yoctolib = require('yoctolib');
var YWakeUpSchedule = yoctolib.YWakeUpSchedule;
php require_once('yocto_wakeupschedule.php');
cpp #include "yocto_wakeupschedule.h"
m #import "yocto_wakeupschedule.h"
pas uses yocto_wakeupschedule;
vb yocto_wakeupschedule.vb
cs yocto_wakeupschedule.cs
java import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py from yocto_wakeupschedule import *

```

### Fonction globales

#### yFindWakeUpSchedule(func)

Permet de retrouver un réveil agendé d'après un identifiant donné.

#### yFirstWakeUpSchedule()

Commence l'énumération des réveils agendés accessibles par la librairie.

### Méthodes des objets YWakeUpSchedule

#### wakeupschedule→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### wakeupschedule→get\_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

#### wakeupschedule→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

#### wakeupschedule→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

#### wakeupschedule→get\_friendlyName()

Retourne un identifiant global du réveil agendé au format NOM\_MODULE . NOM\_FONCTION.

#### wakeupschedule→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### wakeupschedule→get\_functionId()

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

#### wakeupschedule→get\_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format SERIAL . FUNCTIONID.

#### wakeupschedule→get\_hours()

Retourne les heures où le réveil est actif..

#### wakeupschedule→get\_logicalName()

Retourne le nom logique du réveil agendé.

#### wakeupschedule→get\_minutes()

Retourne toutes les minutes de chaque heure où le réveil est actif.

#### wakeupschedule→get\_minutesA()

### 3. Reference

Retourne les minutes de l'intervalle 00-29 de chaque heure où le réveil est actif.

#### wakeupschedule→get\_minutesB()

Retourne les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif.

#### wakeupschedule→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupschedule→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupschedule→get\_monthDays()

Retourne les jours du mois où le réveil est actif..

#### wakeupschedule→get\_months()

Retourne les mois où le réveil est actif..

#### wakeupschedule→get\_nextOccurrence()

Retourne la date/heure de la prochaine occurrence de réveil

#### wakeupschedule→get\_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### wakeupschedule→get\_weekDays()

Retourne les jours de la semaine où le réveil est actif..

#### wakeupschedule→isOnline()

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

#### wakeupschedule→isOnline\_async(callback, context)

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

#### wakeupschedule→load(msValidity)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

#### wakeupschedule→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

#### wakeupschedule→nextWakeUpSchedule()

Continue l'énumération des réveils agendés commencée à l'aide de yFirstWakeUpSchedule().

#### wakeupschedule→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### wakeupschedule→set\_hours(newval, newval)

Modifie les heures où un réveil doit avoir lieu

#### wakeupschedule→set\_logicalName(newval)

Modifie le nom logique du réveil agendé.

#### wakeupschedule→set\_minutes(bitmap)

Modifie toutes les minutes où un réveil doit avoir lieu

#### wakeupschedule→set\_minutesA(newval, newval)

Modifie les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

#### wakeupschedule→set\_minutesB(newval)

Modifie les minutes de l'intervalle 30-59 où un réveil doit avoir lieu.

#### wakeupschedule→set\_monthDays(newval, newval)

Modifie les jours du mois où un réveil doit avoir lieu

#### wakeupschedule→set\_months(newval, newval)

Modifie les mois où un réveil doit avoir lieu

#### wakeupschedule→set\_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**wakeupschedule→set\_weekDays(newval, newval)**

Modifie les jours de la semaine où un réveil doit avoir lieu

**wakeupschedule→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YWakeUpSchedule.FindWakeUpSchedule() yFindWakeUpSchedule()yFindWakeUpSchedule()

### YWakeUpSchedule

Permet de retrouver un réveil agendé d'après un identifiant donné.

**YWakeUpSchedule\* yFindWakeUpSchedule( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le réveil agendé soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpSchedule.isOnline()` pour tester si le réveil agendé est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

#### Paramètres :

**func** une chaîne de caractères qui référence le réveil agendé sans ambiguïté

#### Retourne :

un objet de classe `YWakeUpSchedule` qui permet ensuite de contrôler le réveil agendé.

**YWakeUpSchedule.FirstWakeUpSchedule()****yFirstWakeUpSchedule()yFirstWakeUpSchedule( )****YWakeUpSchedule**

Commence l'énumération des réveils agendés accessibles par la librairie.

**YWakeUpSchedule\* yFirstWakeUpSchedule( )**

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule()` pour itérer sur les autres réveils agendés.

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule`, correspondant au premier réveil agendé accessible en ligne, ou `null` si il n'y a pas de réveils agendés disponibles.

**wakeupschedule→describe()**  
**wakeupschedule→  
describe( )****YWakeUpSchedule**

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le réveil agendé (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

wakeupschedule→get\_advertisedValue()  
wakeupschedule→advertisedValue()  
wakeupschedule→get\_advertisedValue( )

YWakeUpSchedule

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

string get\_advertisedValue( )

**Retourne :**

une chaîne de caractères représentant la valeur courante du réveil agendé (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

wakeupschedule→get\_errorMessage()

**YWakeUpSchedule**

wakeupschedule→errorMessage()wakeupschedule  
→get\_errorMessage( )

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

string **get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

**wakeupschedule→get\_errorType()****YWakeUpSchedule****wakeupschedule→errorType()wakeupschedule→  
get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

wakeupschedule→get\_friendlyName()

**YWakeUpSchedule**

wakeupschedule→friendlyName()wakeupschedule→  
get\_friendlyName( )

---

Retourne un identifiant global du réveil agendé au format NOM\_MODULE . NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du réveil agendé si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du réveil agendé (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le réveil agendé en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

wakeupschedule→get\_functionDescriptor()

YWakeUpSchedule

wakeupschedule→functionDescriptor()

wakeupschedule→get\_functionDescriptor( )

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

wakeupschedule→get\_functionId()

**YWakeUpSchedule**

wakeupschedule→functionId()wakeupschedule→

get\_functionId( )

---

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

string **get\_functionId( )**

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le réveil agendé (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

wakeupschedule→get.hardwareId()

YWakeUpSchedule

wakeupschedule→hardwareId()wakeupschedule→  
get.hardwareId( )

Retourne l'identifiant matériel unique du réveil agendé au format SERIAL.FUNCTIONID.

**string get.hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du réveil agendé (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le réveil agendé (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

wakeupschedule→get\_hours()

YWakeUpSchedule

wakeupschedule→hours()wakeupschedule→  
get\_hours( )

---

Retourne les heures où le réveil est actif..

int get\_hours( )

**Retourne :**

un entier représentant les heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_HOURS\_INVALID.

---

wakeupschedule→get_logicalName()	YWakeUpSchedule
wakeupschedule→logicalName()wakeupschedule→ get_logicalName( )	

---

Retourne le nom logique du réveil agendé.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du réveil agendé. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

wakeupschedule→get\_minutes()

**YWakeUpSchedule**

wakeupschedule→minutes()wakeupschedule→  
get\_minutes( )

---

Retourne toutes les minutes de chaque heure où le réveil est actif.

s64 get\_minutes( )

**wakeupschedule→get\_minutesA()****YWakeUpSchedule****wakeupschedule→minutesA()wakeupschedule→  
get\_minutesA( )**

---

Retourne les minutes de l'intervalle 00-29 de chaque heures où le réveil est actif.

```
int get_minutesA( )
```

**Retourne :**

un entier représentant les minutes de l'intervalle 00-29 de chaque heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MINUTESA\_INVALID.

wakeupschedule→get\_minutesB()

YWakeUpSchedule

wakeupschedule→minutesB()wakeupschedule→

get\_minutesB()

---

Retourne les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif.

```
int get_minutesB()
```

**Retourne :**

un entier représentant les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MINUTESB\_INVALID.

wakeupschedule→get\_module()

YWakeUpSchedule

wakeupschedule→module()wakeupschedule→  
get\_module()

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

wakeupschedule→get\_monthDays()

YWakeUpSchedule

wakeupschedule→monthDays()wakeupschedule→

get\_monthDays( )

---

Retourne les jours du mois où le réveil est actif..

**int get\_monthDays( )**

**Retourne :**

un entier représentant les jours du mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MONTHDAYS\_INVALID.

wakeupschedule→get\_months()

YWakeUpSchedule

wakeupschedule→months()wakeupschedule→  
get\_months( )

---

Retourne les mois où le réveil est actif..

```
int get_months( )
```

**Retourne :**

un entier représentant les mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MONTHS\_INVALID.

wakeupschedule→get\_nextOccurence()  
wakeupschedule→nextOccurence()wakeupschedule  
→get\_nextOccurence( )

YWakeUpSchedule

Retourne la date/heure de la prochaine occurence de réveil

s64 get\_nextOccurence( )

**Retourne :**

un entier représentant la date/heure de la prochaine occurence de réveil

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTOCCURENCE\_INVALID.

**wakeupschedule→get(userData())****YWakeUpSchedule****wakeupschedule→userData()wakeupschedule→  
get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * get(userData( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

wakeupschedule→get\_weekDays()

YWakeUpSchedule

wakeupschedule→weekDays()wakeupschedule→

get\_weekDays( )

---

Retourne les jours de la semaine où le réveil est actif..

```
int get_weekDays( )
```

**Retourne :**

un entier représentant les jours de la semaine où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_WEEKDAYS\_INVALID.

wakeupschedule→**isOnline()** wakeupschedule→  
**isOnline( )**

YWakeUpSchedule

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

bool **isOnline( )**

Si les valeurs des attributs en cache du réveil agendé sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le réveil agendé est joignable, false sinon

**wakeupschedule→load()****YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**nextWakeUpSchedule()**  
wakeupschedule→**nextWakeUpSchedule()**

**YWakeUpSchedule**

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

`YWakeUpSchedule * nextWakeUpSchedule( )`

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wakeupschedule→registerValueCallback()****YWakeUpSchedule****wakeupschedule→registerValueCallback( )**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YWakeUpScheduleValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wakeupschedule→set\_hours()

YWakeUpSchedule

wakeupschedule→setHours()wakeupschedule→  
set\_hours( )

Modifie les heures où un réveil doit avoir lieu

int set\_hours( int newval)

**Paramètres :**

**newval** un entier représentant les heures où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_logicalName()  
wakeupschedule→setLogicalName()  
wakeupschedule→set\_logicalName( )

YWakeUpSchedule

Modifie le nom logique du réveil agendé.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du réveil agendé.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_minutes()**

**YWakeUpSchedule**

**wakeupschedule→setMinutes()wakeupschedule→  
set\_minutes()**

Modifie toutes les minutes où un réveil doit avoir lieu

**int set\_minutes( s64 bitmap)**

**Paramètres :**

**bitmap** Minutes 00-59 de chaque heure où le réveil est actif.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_minutesA()

YWakeUpSchedule

wakeupschedule→setMinutesA()wakeupschedule→  
set\_minutesA( )

Modifie les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

int set\_minutesA( int newval)

**Paramètres :**

**newval** un entier représentant les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_minutesB()

YWakeUpSchedule

wakeupschedule→setMinutesB()wakeupschedule→  
set\_minutesB( )

---

Modifie les minutes de l'intervalle 30-59 où un réveil doit avoir lieu.

int set\_minutesB( int newval)

**Paramètres :**

**newval** un entier représentant les minutes de l'intervalle 30-59 où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_monthDays()	YWakeUpSchedule
wakeupschedule→setMonthDays() wakeupschedule →set_monthDays( )	

Modifie les jours du mois où un réveil doit avoir lieu

```
int set_monthDays( int newval)
```

**Paramètres :**

**newval** un entier représentant les jours du mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_months()

YWakeUpSchedule

wakeupschedule→setMonths()wakeupschedule→  
set\_months( )

Modifie les mois où un réveil doit avoir lieu

int set\_months( int newval)

**Paramètres :**

**newval** un entier représentant les mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set(userData)

YWakeUpSchedule

wakeupschedule→setUserData()wakeupschedule→  
set(userData)

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

void **set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

---

wakeupschedule→set_weekDays()	YWakeUpSchedule
wakeupschedule→setWeekDays()wakeupschedule	
→set_weekDays( )	

Modifie les jours de la semaine où un réveil doit avoir lieu

int **set\_weekDays( int newval)**

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.45. Interface de la fonction Watchdog

La fonction WatchDog est gérée comme un relais qui couperait brièvement l'alimentation d'un appareil après un d'attente temps donné afin de provoquer une réinitialisation complète de cet appareil. Il suffit d'appeler le watchdog à intervalle régulier pour l'empêcher de provoquer la réinitialisation. Le watchdog peut aussi être piloté directement à l'aide des méthodes *pulse* et *delayedpulse* pour éteindre un appareil pendant un temps donné.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_watchdog.js'></script>
nodejs var yoctolib = require('yoctolib');
var YWatchdog = yoctolib.YWatchdog;
php require_once('yocto_watchdog.php');
cpp #include "yocto_watchdog.h"
m #import "yocto_watchdog.h"
pas uses yocto_watchdog;
vb yocto_watchdog.vb
cs yocto_watchdog.cs
java import com.yoctopuce.YoctoAPI.YWatchdog;
py from yocto_watchdog import *

```

### Fonction globales

#### **yFindWatchdog(func)**

Permet de retrouver un watchdog d'après un identifiant donné.

#### **yFirstWatchdog()**

Commence l'énumération des watchdog accessibles par la librairie.

### Méthodes des objets **YWatchdog**

#### **watchdog→delayedPulse(ms\_delay, ms\_duration)**

Pré-programme une impulsion

#### **watchdog→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **watchdog→get\_advertisedValue()**

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

#### **watchdog→get\_autoStart()**

Retourne l'état du watchdog à la mise sous tension du module.

#### **watchdog→get\_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

#### **watchdog→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

#### **watchdog→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

#### **watchdog→get\_friendlyName()**

Retourne un identifiant global du watchdog au format NOM\_MODULE . NOM\_FONCTION.

#### **watchdog→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **watchdog→get\_functionId()**

Retourne l'identifiant matériel du watchdog, sans référence au module.

#### **watchdog->get\_hardwareId()**

Retourne l'identifiant matériel unique du watchdog au format SERIAL.FUNCTIONID.

#### **watchdog->get\_logicalName()**

Retourne le nom logique du watchdog.

#### **watchdog->get\_maxTimeOnStateA()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

#### **watchdog->get\_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

#### **watchdog->get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **watchdog->get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **watchdog->get\_output()**

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

#### **watchdog->get\_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

#### **watchdog->get\_running()**

Retourne l'état du watchdog.

#### **watchdog->get\_state()**

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

#### **watchdog->get\_stateAtPowerOn()**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

#### **watchdog->get\_triggerDelay()**

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

#### **watchdog->get\_triggerDuration()**

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

#### **watchdog->get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **watchdog->isOnline()**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

#### **watchdog->isOnline\_async(callback, context)**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

#### **watchdog->load(msValidity)**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

#### **watchdog->load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

#### **watchdog->nextWatchdog()**

Continue l'énumération des watchdog commencée à l'aide de yFirstWatchdog( ).

#### **watchdog->pulse(ms\_duration)**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

**watchdog→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**watchdog→resetWatchdog()**

Réinitialise le WatchDog.

**watchdog→set\_autoStart(newval)**

Modifie l'état du watching au démarrage du module.

**watchdog→set\_logicalName(newval)**

Modifie le nom logique du watchdog.

**watchdog→set\_maxTimeOnStateA(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**watchdog→set\_maxTimeOnStateB(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

**watchdog→set\_output(newval)**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

**watchdog→set\_running(newval)**

Modifie manuellement l'état de fonctionnement du watchdog.

**watchdog→set\_state(newval)**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

**watchdog→set\_stateAtPowerOn(newval)**

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

**watchdog→set\_triggerDelay(newval)**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

**watchdog→set\_triggerDuration(newval)**

Modifie la durée des resets générés par le watchdog, en millisecondes.

**watchdog→set(userData,data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**watchdog→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YWatchdog.FindWatchdog()****yFindWatchdog()yFindWatchdog( )****YWatchdog**

Permet de retrouver un watchdog d'après un identifiant donné.

**YWatchdog\* yFindWatchdog( const string& func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le watchdog soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWatchdog.isOnline()` pour tester si le watchdog est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le watchdog sans ambiguïté

**Retourne :**

un objet de classe `YWatchdog` qui permet ensuite de contrôler le watchdog.

## **YWatchdog.FirstWatchdog() yFirstWatchdog()yFirstWatchdog( )**

---

**YWatchdog**

Commence l'énumération des watchdog accessibles par la librairie.

**YWatchdog\* yFirstWatchdog( )**

Utiliser la fonction `YWatchdog.nextWatchdog()` pour itérer sur les autres watchdog.

**Retourne :**

un pointeur sur un objet `YWatchdog`, correspondant au premier watchdog accessible en ligne, ou `null` si il n'y a pas de watchdog disponibles.

**watchdog→delayedPulse()**  
**watchdog→delayedPulse( )****YWatchdog**

Pré-programme une impulsion

**int delayedPulse( int ms\_delay, int ms\_duration)****Paramètres :****ms\_delay**      délai d'attente avant l'impulsion, en millisecondes**ms\_duration** durée de l'impulsion, en millisecondes**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog->describe()****YWatchdog**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
string describe( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le watchdog (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**watchdog→get\_advertisedValue()**  
**watchdog→advertisedValue()watchdog→**  
**get\_advertisedValue( )**

**YWatchdog**

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du watchdog (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**watchdog→get\_autoStart()** YWatchdog  
**watchdog→autoStart()watchdog→**  
**get\_autoStart( )**

---

Retourne l'état du watchdog à la mise sous tension du module.

**Y\_AUTOSTART\_enum get\_autoStart( )**

**Retourne :**

soit Y\_AUTOSTART\_OFF, soit Y\_AUTOSTART\_ON, selon l'état du watchdog à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_AUTOSTART\_INVALID.

**watchdog→get\_countdown()**  
**watchdog→countdown()watchdog→**  
**get\_countdown( )**

**YWatchdog**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

s64 **get\_countdown( )**

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne Y\_COUNTDOWN\_INVALID.

**watchdog→get\_errorMessage()**  
**watchdog→errorMessage()** watchdog→  
**get\_errorMessage( )**

---

**YWatchdog**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

---

**watchdog→get\_errorType()**  
**watchdog→errorType()watchdog→**  
**get\_errorType( )**

**YWatchdog**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

**watchdog→get\_friendlyName()**  
**watchdog→friendlyName()**  
**watchdog→get\_friendlyName( )**

---

**YWatchdog**

Retourne un identifiant global du watchdog au format NOM\_MODULE.NOM\_FONCTION.

**string get\_friendlyName( )**

Le chaîne renvoyée utilise soit les noms logiques du module et du watchdog si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du watchdog (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le watchdog en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

---

<b>watchdog→get_functionDescriptor()</b>	<b>YWatchdog</b>
<b>watchdog→functionDescriptor()watchdog→</b>	
<b>get_functionDescriptor( )</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>watchdog→get_functionId()</b>	<b>YWatchdog</b>
<b>watchdog→functionId()watchdog→</b>	
<b>get_functionId( )</b>	

---

Retourne l'identifiant matériel du watchdog, sans référence au module.

```
string get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

---

<b>watchdog→get_hardwareId()</b>	<b>YWatchdog</b>
<b>watchdog→hardwareId()watchdog→ get_hardwareId( )</b>	

---

Retourne l'identifiant matériel unique du watchdog au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du watchdog (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**watchdog→get\_logicalName()**  
**watchdog→logicalName()** watchdog→  
**get\_logicalName( )**

---

**YWatchdog**

Retourne le nom logique du watchdog.

string **get\_logicalName( )**

**Retourne :**

une chaîne de caractères représentant le nom logique du watchdog. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**watchdog→get\_maxTimeOnStateA()****YWatchdog****watchdog→maxTimeOnStateA()watchdog→  
get\_maxTimeOnStateA( )**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**s64 get\_maxTimeOnStateA( )**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEA\_INVALID.

---

<b>watchdog→get_maxTimeOnStateB()</b>	<b>YWatchdog</b>
<b>watchdog→maxTimeOnStateB()watchdog→</b>	
<b>get_maxTimeOnStateB( )</b>	

---

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

**s64 get\_maxTimeOnStateB( )**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEB\_INVALID.

**watchdog→get\_module()****YWatchdog****watchdog→module()watchdog→get\_module( )**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule * get_module( )`

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**watchdog→get\_output()**

**YWatchdog**

**watchdog→output()watchdog→get\_output( )**

---

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

[Y\\_OUTPUT\\_enum get\\_output\( \)](#)

**Retourne :**

soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUT_INVALID`.

**watchdog→get\_pulseTimer()**  
**watchdog→pulseTimer()watchdog→**  
**get\_pulseTimer( )**

**YWatchdog**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

s64 **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\_PULSE\_TIMER\_INVALID.

**watchdog→get\_running()**

**YWatchdog**

**watchdog→running()watchdog→get\_running()**

---

Retourne l'état du watchdog.

**Y\_RUNNING\_enum get\_running( )**

**Retourne :**

soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon l'état du watchdog

En cas d'erreur, déclenche une exception ou retourne Y\_RUNNING\_INVALID.

**watchdog→get\_state()****YWatchdog****watchdog→state()watchdog→get\_state( )**

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

**Y\_STATE\_enum get\_state( )****Retourne :**

soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y\_STATE\_INVALID.

**watchdog→get\_stateAtPowerOn()** YWatchdog  
**watchdog→stateAtPowerOn()** **watchdog→get\_stateAtPowerOn( )**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

**Y\_STATEATPOWERON\_enum get\_stateAtPowerOn( )**

**Retourne :**

une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B représentant l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y\_STATEATPOWERON\_INVALID.

**watchdog→get\_triggerDelay()**  
**watchdog→triggerDelay()watchdog→**  
**get\_triggerDelay()**

**YWatchdog**

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

s64 **get\_triggerDelay( )**

**Retourne :**

un entier représentant le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_TRIGGERDELAY\_INVALID.

**watchdog→get\_triggerDuration()**  
**watchdog→triggerDuration()watchdog→**  
**get\_triggerDuration( )**

---

**YWatchdog**

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

**s64 get\_triggerDuration( )**

**Retourne :**

un entier représentant la durée d'un reset généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_TRIGGER\_DURATION\_INVALID.

**watchdog→get(userData)****YWatchdog****watchdog→userData()watchdog→get(userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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.

**watchdog→isOnline()****YWatchdog**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache du watchdog sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le watchdog est joignable, false sinon

**watchdog→load()****YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→nextWatchdog()watchdog→  
nextWatchdog( )**

---

**YWatchdog**

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

**YWatchdog \* nextWatchdog( )**

**Retourne :**

un pointeur sur un objet `YWatchdog` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**watchdog→pulse()****YWatchdog**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
int pulse( int ms_duration)
```

**Paramètres :**

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→registerValueCallback()**  
**YWatchdog registerValueCallback( )****YWatchdog**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YWatchdogValueCallback 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 appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**watchdog→resetWatchdog()watchdog→  
resetWatchdog( )****YWatchdog**

Réinitialise le WatchDog.

```
int resetWatchdog( )
```

Quand le watchdog est en fonctionnement cette fonction doit être appelée à interval régulier, pour empêcher que le watdog ne se déclenche

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_autoStart()****YWatchdog****watchdog→setAutoStart()watchdog→  
set\_autoStart( )**

Modifie l'état du watching au démarrage du module.

```
int set_autoStart( Y_AUTOSTART_enum newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watching au démarrage du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_logicalName()**  
**watchdog→setLogicalName()****watchdog→set\_logicalName( )**

**YWatchdog**

Modifie le nom logique du watchdog.

**int set\_logicalName( const string& newval)**

Vous pouvez utiliser **yCheckLogicalName( )** pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode **saveToFlash( )** du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du watchdog.

**Retourne :**

**YAPI\_SUCCESS** si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>watchdog→set_maxTimeOnStateA()</b>	<b>YWatchdog</b>
<b>watchdog→setMaxTimeOnStateA()</b>	<b>watchdog→</b>
<b>set_maxTimeOnStateA( )</b>	

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
int set_maxTimeOnStateA( s64 newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_maxTimeOnStateB()**  
**watchdog→setMaxTimeOnStateB()** watchdog→  
**set\_maxTimeOnStateB( )**

**YWatchdog**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

int **set\_maxTimeOnStateB( s64 newval)**

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_output()**

**YWatchdog**

**watchdog→setOutput()watchdog→set\_output( )**

---

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
int set_output( Y_OUTPUT_enum newval)
```

**Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_running()****YWatchdog****watchdog→setRunning()watchdog→  
set\_running()**

Modifie manuellement l'état de fonctionnement du watchdog.

```
int set_running( Y_RUNNING_enum newval)
```

**Paramètres :**

**newval** soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon manuellement l'état de fonctionnement du watchdog

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **watchdog→set\_state()**

**YWatchdog**

## **watchdog→setState()watchdog→set\_state( )**

---

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
int set_state( Y_STATE_enum newval)
```

**Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>watchdog→set_stateAtPowerOn()</b>	<b>YWatchdog</b>
<b>watchdog→setStateAtPowerOn()watchdog→</b> <b>set_stateAtPowerOn( )</b>	

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
int set_stateAtPowerOn( Y_STATEATPOWERON_enum newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDelay()**  
**watchdog→setTriggerDelay()**  
**watchdog→set\_triggerDelay( )**

**YWatchdog**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

**int set\_triggerDelay( s64 newval)**

**Paramètres :**

**newval** un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDuration()****YWatchdog****watchdog→setTriggerDuration()watchdog→  
set\_triggerDuration( )**

---

Modifie la durée des resets générés par le watchdog, en millisecondes.

```
int set_triggerDuration( s64 newval)
```

**Paramètres :**

**newval** un entier représentant la durée des resets générés par le watchdog, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set(userData())**

**YWatchdog**

**watchdog→setUserData()watchdog→  
set(userData())**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
void setUserData( void* data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.46. Interface de la fonction Wireless

La fonction YWireless permet de configurer et de contrôler la configuration du réseau sans fil sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wireless.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YWireless = yoctolib.YWireless;
php	require_once('yocto_wireless.php');
cpp	#include "yocto_wireless.h"
m	#import "yocto_wireless.h"
pas	uses yocto_wireless;
vb	yocto_wireless.vb
cs	yocto_wireless.cs
java	import com.yoctopuce.YoctoAPI.YWireless;
py	from yocto_wireless import *

### 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 de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### 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 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

#### wireless→get\_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

#### wireless→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

#### wireless→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

#### wireless→get\_friendlyName()

Retourne un identifiant global de l'interface réseau sans fil au format NOM\_MODULE . NOM\_FONCTION.

#### wireless→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### wireless→get\_functionId()

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

#### wireless→get\_hardwareId()

### 3. Reference

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

#### wireless→get\_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

#### wireless→get\_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

#### wireless→get\_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

#### wireless→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wireless→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wireless→get\_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

#### wireless→get\_ssid()

Retourne le nom (SSID) du réseau sans-fil sélectionné.

#### wireless→get\_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### wireless→isOnline()

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

#### wireless→isOnline\_async(callback, context)

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

#### wireless→joinNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

#### wireless→load(msValidity)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

#### wireless→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

#### wireless→nextWireless()

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de yFirstWireless( ).

#### wireless→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### wireless→set\_logicalName(newval)

Modifie le nom logique de l'interface réseau sans fil.

#### wireless→set(userData)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### wireless→wait\_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YWireless.FindWireless()****YWireless****yFindWireless()yFindWireless( )**

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

**YWireless\* yFindWireless( string func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau sans fil soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWireless.isOnline()` pour tester si l'interface réseau sans fil est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'interface réseau sans fil sans ambiguïté

**Retourne :**

un objet de classe `YWireless` qui permet ensuite de contrôler l'interface réseau sans fil.

## **YWireless.FirstWireless() yFirstWireless()yFirstWireless()**

---

**YWireless**

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

**YWireless\* yFirstWireless( )**

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

**Retourne :**

un pointeur sur un objet `YWireless`, correspondant à la première interface réseau sans fil accessible en ligne, ou `null` si il n'y a pas de interfaces réseau sans fil disponibles.

**wireless→adhocNetwork()wireless→adhocNetwork( )****YWireless**

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

```
int adhocNetwork( string ssid, string securityKey)
```

Si une clef d'accès est spécifiée, le réseau sera protégé par une sécurité WEP128 (l'utilisation de WPA n'est pas standardisée en mode ad-hoc). N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à créer

**securityKey** clé d'accès de réseau, sous forme de chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→describe(wireless→describe( )****YWireless**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**string describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'interface réseau sans fil (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**wireless→get\_advertisedValue()**  
**wireless→advertisedValue()wireless→**  
**get\_advertisedValue( )**

**YWireless**

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

**string get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wireless→get\_channel()**

**YWireless**

**wireless→channel()wireless→get\_channel()**

---

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

**int get\_channel( )**

**Retourne :**

un entier représentant le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé

En cas d'erreur, déclenche une exception ou retourne Y\_CHANNEL\_INVALID.

---

<b>wireless→get_detectedWlans()</b>	<b>YWireless</b>
<b>wireless→detectedWlans()wireless→</b>	
<b>get_detectedWlans( )</b>	

---

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

```
vector<YWlanRecord> get_detectedWlans( )
```

La liste n'est pas mise à jour quand le module est déjà connecté à un accès sans fil (mode "infrastructure"). Pour forcer la détection des réseaux sans fil, il faut appeler addhocNetwork( ) pour se déconnecter du réseau actuel. L'appelant est responsable de la désallocation de la liste renvoyée.

**Retourne :**

une liste d'objets YWlanRecord, contenant le SSID, le canal, la qualité du signal, et l'algorithme de sécurité utilisé par le réseau sans-fil

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**wireless→get\_errorMessage()**  
**wireless→errorMessage()wireless→**  
**get\_errorMessage( )**

---

**YWireless**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

**string get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

**wireless→get\_errorType()**

**YWireless**

**wireless→errorType()wireless→get\_errorType( )**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

**YRETCODE get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

wireless→get\_friendlyName()  
wireless→friendlyName()wireless→  
get\_friendlyName( )

YWireless

Retourne un identifiant global de l'interface réseau sans fil au format NOM\_MODULE.NOM\_FONCTION.

string get\_friendlyName( )

Le chaîne renvoyée utilise soit les noms logiques du module et de l'interface réseau sans fil si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau sans fil (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**wireless→get\_functionDescriptor()**  
**wireless→functionDescriptor()wireless→**  
**get\_functionDescriptor( )**

**YWireless**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**YFUN\_DESCR 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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**wireless→get\_functionId()**

**YWireless**

**wireless→functionId()wireless→**

**get\_functionId( )**

---

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

**string get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**wireless→get\_hardwareId()**  
**wireless→hardwareId()wireless→**  
**get\_hardwareId( )**

**YWireless**

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

**string get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau sans fil (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**wireless→get\_linkQuality()**

**YWireless**

**wireless→linkQuality()wireless→  
get\_linkQuality()**

---

Retourne la qualité de la connection, exprimée en pourcents.

**int 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()**  
**wireless→logicalName()wireless→**  
**get\_logicalName( )**

**YWireless**

Retourne le nom logique de l'interface réseau sans fil.

```
string get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau sans fil. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**wireless→get\_message()**

**YWireless**

**wireless→message()wireless→get\_message( )**

---

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

**string get\_message( )**

**Retourne :**

une chaîne de caractères représentant le dernier message de diagnostic de l'interface au réseau sans fil

En cas d'erreur, déclenche une exception ou retourne Y\_MESSAGE\_INVALID.

**wireless→get\_module()****YWireless****wireless→module()wireless→get\_module( )**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**YModule \* get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

**wireless→get\_security()**

**YWireless**

**wireless→security()wireless→get\_security()**

---

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

**Y\_SECURITY\_enum get\_security( )**

**Retourne :**

une valeur parmi Y\_SECURITY\_UNKNOWN, Y\_SECURITY\_OPEN, Y\_SECURITY\_WEP, Y\_SECURITY\_WPA et Y\_SECURITY\_WPA2 représentant l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y\_SECURITY\_INVALID.

**wireless→get\_ssid()****YWireless****wireless→ssid()wireless→get\_ssid( )**

Retourne le nom (SSID) du réseau sans-fil sélectionné.

```
string get_ssid( )
```

**Retourne :**

une chaîne de caractères représentant le nom (SSID) du réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y\_SSID\_INVALID.

**wireless→get(userData)**

**YWireless**

**wireless→userData(wireless→get(userData))**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
void * 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()|wireless→isOnline()****YWireless**

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

**bool isOnline( )**

Si les valeurs des attributs en cache de l'interface réseau sans fil sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'interface réseau sans fil est joignable, false sinon

**wireless→joinNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

```
int joinNetwork( string ssid, string securityKey)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à utiliser

**securityKey** clé d'accès au réseau, sous forme de chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→load()wireless→load( )****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

**YRETCODE load( int msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→nextWireless()wireless→**  
**nextWireless( )**

---

**YWireless**

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

`YWireless * nextWireless( )`

**Retourne :**

un pointeur sur un objet `YWireless` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wireless→registerValueCallback()**  
**wireless→registerValueCallback( )****YWireless**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( YWirelessValueCallback 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 appellés 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.

<b>wireless→set_logicalName()</b>	<b>YWireless</b>
<b>wireless→setLogicalName()wireless→ set_logicalName( )</b>	

Modifie le nom logique de l'interface réseau sans fil.

```
int set_logicalName( const string& newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→set(userData)**  
**wireless→setUserData()wireless→**  
**set(userData( )**

**YWireless**

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).

**void set(userData( void\* data)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser



# Index

## A

Accelerometer 33  
adhocNetwork, YWireless 1560  
Alimentation 443  
AnButton 75

## B

Blueprint 12  
Brute 305

## C

C++ 3, 8  
calibrate, YLightSensor 688  
calibrateFromPoints, YAccelerometer 37  
calibrateFromPoints, YCarbonDioxide 117  
calibrateFromPoints, YCompass 185  
calibrateFromPoints, YCurrent 225  
calibrateFromPoints, YGenericSensor 500  
calibrateFromPoints, YGyro 546  
calibrateFromPoints, YHumidity 622  
calibrateFromPoints, YLightSensor 689  
calibrateFromPoints, YMagnetometer 728  
calibrateFromPoints, YPower 899  
calibrateFromPoints, YPressure 942  
calibrateFromPoints, YQt 1042  
calibrateFromPoints,YSensor 1180  
calibrateFromPoints, YTemperature 1254  
calibrateFromPoints, YTilt 1295  
calibrateFromPoints, YVoc 1334  
calibrateFromPoints, YVoltage 1373  
callbackLogin, YNetwork 820  
cancel3DCalibration, YRefFrame 1108  
CarbonDioxide 113  
CheckLogicalName, YAPI 14  
clear, YDisplayLayer 412  
clearConsole, YDisplayLayer 413  
ColorLed 152  
Compass 181  
Configuration 1104  
consoleOut, YDisplayLayer 414  
Contrôle 3, 5, 443, 772, 872  
copyLayerContent, YDisplay 368  
Current 221

## D

DataLogger 260  
delayedPulse, YDigitalIO 324  
delayedPulse, YRelay 1144  
delayedPulse, YWatchdog 1516  
describe, YAccelerometer 38  
describe, YAnButton 79  
describe, YCarbonDioxide 118

describe, YColorLed 155  
describe, YCompass 186  
describe, YCurrent 226  
describe, YDataLogger 263  
describe, YDigitalIO 325  
describe, YDisplay 369  
describe, YDualPower 446  
describe, YFiles 471  
describe, YGenericSensor 501  
describe, YGyro 547  
describe, YHubPort 596  
describe, YHumidity 623  
describe, YLed 660  
describe, YLightSensor 690  
describe, YMagnetometer 729  
describe, YModule 776  
describe, YNetwork 821  
describe, YOsControl 875  
describe, YPower 900  
describe, YPressure 943  
describe, YPwmOutput 981  
describe, YPwmPowerSource 1018  
describe, YQt 1043  
describe, YRealTimeClock 1080  
describe, YRefFrame 1109  
describe, YRelay 1145  
describe, YSensor 1181  
describe, YServo 1219  
describe, YTemperature 1255  
describe, YTilt 1296  
describe, YVoc 1335  
describe, YVoltage 1374  
describe, YVSource 1411  
describe, YWakeUpMonitor 1444  
describe, YWakeUpSchedule 1479  
describe, YWatchdog 1517  
describe, YWireless 1561  
DigitalIO 320  
DisableExceptions, YAPI 15  
Display 364  
DisplayLayer 411  
Données 291, 293, 305  
download, YFiles 472  
download, YModule 777  
drawBar, YDisplayLayer 415  
drawBitmap, YDisplayLayer 416  
drawCircle, YDisplayLayer 417  
drawDisc, YDisplayLayer 418  
drawImage, YDisplayLayer 419  
drawPixel, YDisplayLayer 420  
drawRect, YDisplayLayer 421  
drawText, YDisplayLayer 422  
dutyCycleMove, YPwmOutput 982

## E

EnableExceptions, YAPI 16  
Enregistrées 293, 305  
Erreurs 7

## F

fade, YDisplay 370  
Files 468  
FindAccelerometer, YAccelerometer 35  
FindAnButton, YAnButton 77  
FindCarbonDioxide, YCarbonDioxide 115  
FindColorLed, YColorLed 153  
FindCompass, YCompass 183  
FindCurrent, YCurrent 223  
FindDataLogger, YDataLogger 261  
FindDigitalIO, YDigitalIO 322  
FindDisplay, YDisplay 366  
FindDualPower, YDualPower 444  
FindFiles, YFiles 469  
FindGenericSensor, YGenericSensor 498  
FindGyro, YGyro 544  
FindHubPort, YHubPort 594  
FindHumidity, YHumidity 620  
FindLed, YLed 658  
FindLightSensor, YLightSensor 686  
FindMagnetometer, YMagnetometer 726  
FindModule, YModule 774  
FindNetwork, YNetwork 818  
FindOsControl, YOsControl 873  
FindPower, YPower 897  
FindPressure, YPressure 940  
FindPwmOutput, YPwmOutput 979  
FindPwmPowerSource, YPwmPowerSource 1016  
FindQt, YQt 1040  
FindRealTimeClock, YRealTimeClock 1078  
FindRefFrame, YRefFrame 1106  
FindRelay, YRelay 1142  
FindSensor, YSensor 1178  
FindServo,YServo 1217  
FindTemperature, YTemperature 1252  
FindTilt, YTilt 1293  
FindVoc, YVoc 1332  
FindVoltage, YVoltage 1371  
FindVSource, YVSource 1409  
FindWakeUpMonitor, YWakeUpMonitor 1442  
FindWakeUpSchedule, YWakeUpSchedule 1477  
FindWatchdog, YWatchdog 1514  
FindWireless, YWireless 1558  
FirstAccelerometer, YAccelerometer 36  
FirstAnButton, YAnButton 78  
FirstCarbonDioxide, YCarbonDioxide 116  
FirstColorLed, YColorLed 154  
FirstCompass, YCompass 184  
FirstCurrent, YCurrent 224  
FirstDataLogger, YDataLogger 262  
FirstDigitalIO, YDigitalIO 323  
FirstDisplay, YDisplay 367

FirstDualPower, YDualPower 445

FirstFiles, YFiles 470

FirstGenericSensor, YGenericSensor 499

FirstGyro, YGyro 545

FirstHubPort, YHubPort 595

FirstHumidity, YHumidity 621

FirstLed, YLed 659

FirstLightSensor, YLightSensor 687

FirstMagnetometer, YMagnetometer 727

FirstModule, YModule 775

FirstNetwork, YNetwork 819

FirstOsControl, YOsControl 874

FirstPower, YPower 898

FirstPressure, YPressure 941

FirstPwmOutput, YPwmOutput 980

FirstPwmPowerSource, YPwmPowerSource 1017

FirstQt, YQt 1041

FirstRealTimeClock, YRealTimeClock 1079

FirstRefFrame, YRefFrame 1107

FirstRelay, YRelay 1143

FirstSensor, YSensor 1179

FirstServo, YServo 1218

FirstTemperature, YTemperature 1253

FirstTilt, YTilt 1294

FirstVoc, YVoc 1333

FirstVoltage, YVoltage 1372

FirstVSource, YVSource 1410

FirstWakeUpMonitor, YWakeUpMonitor 1443

FirstWakeUpSchedule, YWakeUpSchedule 1478

FirstWatchdog, YWatchdog 1515

FirstWireless, YWireless 1559

Fonctions 13, 1176

forgetAllDataStreams, YDataLogger 264

format\_fs, YFiles 473

Forme 291

FreeAPI, YAPI 17

functionCount, YModule 778

functionId, YModule 779

functionName, YModule 780

functionValue, YModule 781

## G

GenericSensor 496

get\_3DCalibrationHint, YRefFrame 1110

get\_3DCalibrationLogMsg, YRefFrame 1111

get\_3DCalibrationProgress, YRefFrame 1112

get\_3DCalibrationStage, YRefFrame 1113

get\_3DCalibrationStageProgress, YRefFrame 1114

get\_adminPassword, YNetwork 822

get\_advertisedValue, YAccelerometer 39

get\_advertisedValue, YAnButton 80

get\_advertisedValue, YCarbonDioxide 119

get\_advertisedValue, YColorLed 156

get\_advertisedValue, YCompass 187

get\_advertisedValue, YCurrent 227

get\_advertisedValue, YDataLogger 265

get\_advertisedValue, YDigitalIO 326

get\_advertisedValue, YDisplay 371  
get\_advertisedValue, YDualPower 447  
get\_advertisedValue, YFiles 474  
get\_advertisedValue, YGenericSensor 502  
get\_advertisedValue, YGyro 548  
get\_advertisedValue, YHubPort 597  
get\_advertisedValue, YHumidity 624  
get\_advertisedValue, YLed 661  
get\_advertisedValue, YLightSensor 691  
get\_advertisedValue, YMagnetometer 730  
get\_advertisedValue, YNetwork 823  
get\_advertisedValue, YOsControl 876  
get\_advertisedValue, YPower 901  
get\_advertisedValue, YPressure 944  
get\_advertisedValue, YPwmOutput 983  
get\_advertisedValue, YPwmPowerSource 1019  
get\_advertisedValue, YQt 1044  
get\_advertisedValue, YRealTimeClock 1081  
get\_advertisedValue, YRefFrame 1115  
get\_advertisedValue, YRelay 1146  
get\_advertisedValue, YSensor 1182  
get\_advertisedValue,YServo 1220  
get\_advertisedValue, YTemperature 1256  
get\_advertisedValue, YTilt 1297  
get\_advertisedValue, YVoc 1336  
get\_advertisedValue, YVoltage 1375  
get\_advertisedValue, YVSource 1412  
get\_advertisedValue, YWakeUpMonitor 1445  
get\_advertisedValue, YWakeUpSchedule 1480  
get\_advertisedValue, YWatchdog 1518  
get\_advertisedValue, YWireless 1562  
get\_analogCalibration, YAnButton 81  
get\_autoStart, YDataLogger 266  
get\_autoStart, YWatchdog 1519  
get\_averageValue, YDataStream 306  
get\_averageValue, YMeasure 766  
get\_baudRate, YHubPort 598  
get\_beacon, YModule 782  
get\_bearing, YRefFrame 1116  
get\_bitDirection, YDigitalIO 327  
get\_bitOpenDrain, YDigitalIO 328  
get\_bitPolarity, YDigitalIO 329  
get\_bitState, YDigitalIO 330  
get\_blinking, YLed 662  
get\_brightness, YDisplay 372  
get\_calibratedValue, YAnButton 82  
get\_calibrationMax, YAnButton 83  
get\_calibrationMin, YAnButton 84  
get\_callbackCredentials, YNetwork 824  
get\_callbackEncoding, YNetwork 825  
get\_callbackMaxDelay, YNetwork 826  
get\_callbackMethod, YNetwork 827  
get\_callbackMinDelay, YNetwork 828  
get\_callbackUrl, YNetwork 829  
get\_channel, YWireless 1563  
get\_columnCount, YDataStream 307  
get\_columnNames, YDataStream 308  
get\_cosPhi, YPower 902  
get\_countdown, YRelay 1147  
get\_countdown, YWatchdog 1520  
get\_currentRawValue, YAccelerometer 40  
get\_currentRawValue, YCarbonDioxide 120  
get\_currentRawValue, YCompass 188  
get\_currentRawValue, YCurrent 228  
get\_currentRawValue, YGenericSensor 503  
get\_currentRawValue, YGyro 549  
get\_currentRawValue, YHumidity 625  
get\_currentRawValue, YLightSensor 692  
get\_currentRawValue, YMagnetometer 731  
get\_currentRawValue, YPower 903  
get\_currentRawValue, YPressure 945  
get\_currentRawValue, YQt 1045  
get\_currentRawValue, YSensor 1183  
get\_currentRawValue, YTemperature 1257  
get\_currentRawValue, YTilt 1298  
get\_currentRawValue, YVoc 1337  
get\_currentRawValue, YVoltage 1376  
get\_currentRunIndex, YDataLogger 267  
get\_currentValue, YAccelerometer 41  
get\_currentValue, YCarbonDioxide 121  
get\_currentValue, YCompass 189  
get\_currentValue, YCurrent 229  
get\_currentValue, YGenericSensor 504  
get\_currentValue, YGyro 550  
get\_currentValue, YHumidity 626  
get\_currentValue, YLightSensor 693  
get\_currentValue, YMagnetometer 732  
get\_currentValue, YPower 904  
get\_currentValue, YPressure 946  
get\_currentValue, YQt 1046  
get\_currentValue, YSensor 1184  
get\_currentValue, YTemperature 1258  
get\_currentValue, YTilt 1299  
get\_currentValue, YVoc 1338  
get\_currentValue, YVoltage 1377  
get\_data, YDataStream 309  
get\_dataRows, YDataStream 310  
get\_dataSamplesIntervalMs, YDataStream 311  
get\_dataSets, YDataLogger 268  
get\_dataStreams, YDataLogger 269  
get\_dateTime, YRealTimeClock 1082  
get\_detectedWlans, YWireless 1564  
get\_discoverable, YNetwork 830  
get\_display, YDisplayLayer 423  
get\_displayHeight, YDisplay 373  
get\_displayHeight, YDisplayLayer 424  
get\_displayLayer, YDisplay 374  
get\_displayType, YDisplay 375  
get\_displayWidth, YDisplay 376  
get\_displayWidth, YDisplayLayer 425  
get\_duration, YDataStream 312  
get\_dutyCycle, YPwmOutput 984  
get\_dutyCycleAtPowerOn, YPwmOutput 985  
get\_enabled, YDisplay 377  
get\_enabled, YHubPort 599  
get\_enabled, YPwmOutput 986  
get\_enabled, YServo 1221  
get\_enabledAtPowerOn, YPwmOutput 987

get\_enabledAtPowerOn, YServo 1222  
get\_endTimeUTC, YDataSet 294  
get\_endTimeUTC, YMeasure 767  
get\_errorMessage, YAccelerometer 42  
get\_errorMessage, YAnButton 85  
get\_errorMessage, YCarbonDioxide 122  
get\_errorMessage, YColorLed 157  
get\_errorMessage, YCompass 190  
get\_errorMessage, YCurrent 230  
get\_errorMessage, YDataLogger 270  
get\_errorMessage, YDigitalIO 331  
get\_errorMessage, YDisplay 378  
get\_errorMessage, YDualPower 448  
get\_errorMessage, YFiles 475  
get\_errorMessage, YGenericSensor 505  
get\_errorMessage, YGyro 551  
get\_errorMessage, YHubPort 600  
get\_errorMessage, YHumidity 627  
get\_errorMessage, YLed 663  
get\_errorMessage, YLightSensor 694  
get\_errorMessage, YMagnetometer 733  
get\_errorMessage, YModule 783  
get\_errorMessage, YNetwork 831  
get\_errorMessage, YOsControl 877  
get\_errorMessage, YPower 905  
get\_errorMessage, YPressure 947  
get\_errorMessage, YPwmOutput 988  
get\_errorMessage, YPwmPowerSource 1020  
get\_errorMessage, YQt 1047  
get\_errorMessage, YRealTimeClock 1083  
get\_errorMessage, YRefFrame 1117  
get\_errorMessage, YRelay 1148  
get\_errorMessage, YSensor 1185  
get\_errorMessage, YServo 1223  
get\_errorMessage, YTemperature 1259  
get\_errorMessage, YTilt 1300  
get\_errorMessage, YVoc 1339  
get\_errorMessage, YVoltage 1378  
get\_errorMessage, YVSource 1413  
get\_errorMessage, YWakeUpMonitor 1446  
get\_errorMessage, YWakeUpSchedule 1481  
get\_errorMessage, YWatchdog 1521  
get\_errorMessage, YWireless 1565  
get\_errorType, YAccelerometer 43  
get\_errorType, YAnButton 86  
get\_errorType, YCarbonDioxide 123  
get\_errorType, YColorLed 158  
get\_errorType, YCompass 191  
get\_errorType, YCurrent 231  
get\_errorType, YDataLogger 271  
get\_errorType, YDigitalIO 332  
get\_errorType, YDisplay 379  
get\_errorType, YDualPower 449  
get\_errorType, YFiles 476  
get\_errorType, YGenericSensor 506  
get\_errorType, YGyro 552  
get\_errorType, YHubPort 601  
get\_errorType, YHumidity 628  
get\_errorType, YLed 664  
get\_errorType, YLightSensor 695  
get\_errorType, YMagnetometer 734  
get\_errorType, YModule 784  
get\_errorType, YNetwork 832  
get\_errorType, YOsControl 878  
get\_errorType, YPower 906  
get\_errorType, YPressure 948  
get\_errorType, YPwmOutput 989  
get\_errorType, YPwmPowerSource 1021  
get\_errorType, YQt 1048  
get\_errorType, YRealTimeClock 1084  
get\_errorType, YRefFrame 1118  
get\_errorType, YRelay 1149  
get\_errorType, YSensor 1186  
get\_errorType, YServo 1224  
get\_errorType, YTemperature 1260  
get\_errorType, YTilt 1301  
get\_errorType, YVoc 1340  
get\_errorType, YVoltage 1379  
get\_errorType, YVSource 1414  
get\_errorType, YWakeUpMonitor 1447  
get\_errorType, YWakeUpSchedule 1482  
get\_errorType, YWatchdog 1522  
get\_errorType, YWireless 1566  
get\_extPowerFailure, YVSource 1415  
get\_extVoltage, YDualPower 450  
get\_failure, YVSource 1416  
get\_filesCount, YFiles 477  
get\_firmwareRelease, YModule 785  
get\_freeSpace, YFiles 478  
get\_frequency, YPwmOutput 990  
get\_friendlyName, YAccelerometer 44  
get\_friendlyName, YAnButton 87  
get\_friendlyName, YCarbonDioxide 124  
get\_friendlyName, YColorLed 159  
get\_friendlyName, YCompass 192  
get\_friendlyName, YCurrent 232  
get\_friendlyName, YDataLogger 272  
get\_friendlyName, YDigitalIO 333  
get\_friendlyName, YDisplay 380  
get\_friendlyName, YDualPower 451  
get\_friendlyName, YFiles 479  
get\_friendlyName, YGenericSensor 507  
get\_friendlyName, YGyro 553  
get\_friendlyName, YHubPort 602  
get\_friendlyName, YHumidity 629  
get\_friendlyName, YLed 665  
get\_friendlyName, YLightSensor 696  
get\_friendlyName, YMagnetometer 735  
get\_friendlyName, YNetwork 833  
get\_friendlyName, YOsControl 879  
get\_friendlyName, YPower 907  
get\_friendlyName, YPressure 949  
get\_friendlyName, YPwmOutput 991  
get\_friendlyName, YPwmPowerSource 1022  
get\_friendlyName, YQt 1049  
get\_friendlyName, YRealTimeClock 1085  
get\_friendlyName, YRefFrame 1119  
get\_friendlyName, YRelay 1150

get\_friendlyName, YSensor 1187  
get\_friendlyName,YServo 1225  
get\_friendlyName,YTemperature 1261  
get\_friendlyName,YTilt 1302  
get\_friendlyName,YVoc 1341  
get\_friendlyName,YVoltage 1380  
get\_friendlyName,YVSource 1417  
get\_friendlyName,YWakeUpMonitor 1448  
get\_friendlyName,YWakeUpSchedule 1483  
get\_friendlyName,YWatchdog 1523  
get\_friendlyName,YWireless 1567  
get\_functionDescriptor, YAccelerometer 45  
get\_functionDescriptor, YAnButton 88  
get\_functionDescriptor, YCarbonDioxide 125  
get\_functionDescriptor, YColorLed 160  
get\_functionDescriptor, YCompass 193  
get\_functionDescriptor, YCurrent 233  
get\_functionDescriptor, YDataLogger 273  
get\_functionDescriptor, YDigitalIO 334  
get\_functionDescriptor, YDisplay 381  
get\_functionDescriptor, YDualPower 452  
get\_functionDescriptor, YFiles 480  
get\_functionDescriptor, YGenericSensor 508  
get\_functionDescriptor, YGyro 554  
get\_functionDescriptor, YHubPort 603  
get\_functionDescriptor, YHumidity 630  
get\_functionDescriptor, YLed 666  
get\_functionDescriptor, YLightSensor 697  
get\_functionDescriptor, YMagnetometer 736  
get\_functionDescriptor, YNetwork 834  
get\_functionDescriptor, YOsControl 880  
get\_functionDescriptor, YPower 908  
get\_functionDescriptor, YPressure 950  
get\_functionDescriptor, YPwmOutput 992  
get\_functionDescriptor, YPwmPowerSource 1023  
get\_functionDescriptor, YQt 1050  
get\_functionDescriptor, YRealTimeClock 1086  
get\_functionDescriptor, YRefFrame 1120  
get\_functionDescriptor, YRelay 1151  
get\_functionDescriptor, YSensor 1188  
get\_functionDescriptor, YServo 1226  
get\_functionDescriptor, YTemperature 1262  
get\_functionDescriptor, YTilt 1303  
get\_functionDescriptor, YVoc 1342  
get\_functionDescriptor, YVoltage 1381  
get\_functionDescriptor, YVSource 1418  
get\_functionDescriptor, YWakeUpMonitor 1449  
get\_functionDescriptor, YWakeUpSchedule 1484  
get\_functionDescriptor, YWatchdog 1524  
get\_functionDescriptor, YWireless 1568  
get\_functionId, YAccelerometer 46  
get\_functionId, YAnButton 89  
get\_functionId, YCarbonDioxide 126  
get\_functionId, YColorLed 161  
get\_functionId, YCompass 194  
get\_functionId, YCurrent 234  
get\_functionId, YDataLogger 274  
get\_functionId, YDataSet 295  
get\_functionId, YDigitalIO 335  
get\_functionId, YDisplay 382  
get\_functionId, YDualPower 453  
get\_functionId, YFiles 481  
get\_functionId, YGenericSensor 509  
get\_functionId, YGyro 555  
get\_functionId, YHubPort 604  
get\_functionId, YHumidity 631  
get\_functionId, YLed 667  
get\_functionId, YLightSensor 698  
get\_functionId, YMagnetometer 737  
get\_functionId, YNetwork 835  
get\_functionId, YOsControl 881  
get\_functionId, YPower 909  
get\_functionId, YPressure 951  
get\_functionId, YPwmOutput 993  
get\_functionId, YPwmPowerSource 1024  
get\_functionId, YQt 1051  
get\_functionId, YRealTimeClock 1087  
get\_functionId, YRefFrame 1121  
get\_functionId, YRelay 1152  
get\_functionId, YSensor 1189  
get\_functionId, YServo 1227  
get\_functionId, YTemperature 1263  
get\_functionId, YTilt 1304  
get\_functionId, YVoc 1343  
get\_functionId, YVoltage 1382  
get\_functionId, YVSource 1419  
get\_functionId, YWakeUpMonitor 1450  
get\_functionId, YWakeUpSchedule 1485  
get\_functionId, YWatchdog 1525  
get\_functionId, YWireless 1569  
get\_hardwareId, YAccelerometer 47  
get\_hardwareId, YAnButton 90  
get\_hardwareId, YCarbonDioxide 127  
get\_hardwareId, YColorLed 162  
get\_hardwareId, YCompass 195  
get\_hardwareId, YCurrent 235  
get\_hardwareId, YDataLogger 275  
get\_hardwareId, YDataSet 296  
get\_hardwareId, YDigitalIO 336  
get\_hardwareId, YDisplay 383  
get\_hardwareId, YDualPower 454  
get\_hardwareId, YFiles 482  
get\_hardwareId, YGenericSensor 510  
get\_hardwareId, YGyro 556  
get\_hardwareId, YHubPort 605  
get\_hardwareId, YHumidity 632  
get\_hardwareId, YLed 668  
get\_hardwareId, YLightSensor 699  
get\_hardwareId, YMagnetometer 738  
get\_hardwareId, YModule 786  
get\_hardwareId, YNetwork 836  
get\_hardwareId, YOsControl 882  
get\_hardwareId, YPower 910  
get\_hardwareId, YPressure 952  
get\_hardwareId, YPwmOutput 994  
get\_hardwareId, YPwmPowerSource 1025  
get\_hardwareId, YQt 1052  
get\_hardwareId, YRealTimeClock 1088

get\_hardwareId, YRefFrame 1122  
get\_hardwareId, YRelay 1153  
get\_hardwareId,YSensor 1190  
get\_hardwareId,YServo 1228  
get\_hardwareId,YTemperature 1264  
get\_hardwareId,YTilt 1305  
get\_hardwareId,YVoc 1344  
get\_hardwareId,YVoltage 1383  
get\_hardwareId,YVSource 1420  
get\_hardwareId,YWakeUpMonitor 1451  
get\_hardwareId,YWakeUpSchedule 1486  
get\_hardwareId,YWatchdog 1526  
get\_hardwareId,YWireless 1570  
get\_heading, YGyro 557  
get\_highestValue, YAccelerometer 48  
get\_highestValue, YCarbonDioxide 128  
get\_highestValue, YCompass 196  
get\_highestValue, YCurrent 236  
get\_highestValue, YGenericSensor 511  
get\_highestValue, YGyro 558  
get\_highestValue, YHumidity 633  
get\_highestValue, YLightSensor 700  
get\_highestValue, YMagnetometer 739  
get\_highestValue, YPower 911  
get\_highestValue, YPressure 953  
get\_highestValue, YQt 1053  
get\_highestValue,YSensor 1191  
get\_highestValue,YTemperature 1265  
get\_highestValue, YTilt 1306  
get\_highestValue,YVoc 1345  
get\_highestValue,YVoltage 1384  
get\_hours, YWakeUpSchedule 1487  
get\_hslColor, YColorLed 163  
get\_icon2d, YModule 787  
get\_ipAddress, YNetwork 837  
get\_isPressed, YAnButton 91  
get\_lastLogs, YModule 788  
get\_lastTimePressed, YAnButton 92  
get\_lastTimeReleased, YAnButton 93  
get\_layerCount, YDisplay 384  
get\_layerHeight, YDisplay 385  
get\_layerHeight, YDisplayLayer 426  
get\_layerWidth, YDisplay 386  
get\_layerWidth, YDisplayLayer 427  
get\_linkQuality, YWireless 1571  
get\_list, YFiles 483  
get\_logFrequency, YAccelerometer 49  
get\_logFrequency, YCarbonDioxide 129  
get\_logFrequency, YCompass 197  
get\_logFrequency, YCurrent 237  
get\_logFrequency, YGenericSensor 512  
get\_logFrequency, YGyro 559  
get\_logFrequency, YHumidity 634  
get\_logFrequency, YLightSensor 701  
get\_logFrequency, YMagnetometer 740  
get\_logFrequency, YPower 912  
get\_logFrequency, YPressure 954  
get\_logFrequency, YQt 1054  
get\_logFrequency,YSensor 1192  
get\_logFrequency, YTemperature 1266  
get\_logFrequency, YTilt 1307  
get\_logFrequency, YVoc 1346  
get\_logFrequency, YVoltage 1385  
get\_logicalName, YAccelerometer 50  
get\_logicalName, YAnButton 94  
get\_logicalName, YCarbonDioxide 130  
get\_logicalName, YColorLed 164  
get\_logicalName, YCompass 198  
get\_logicalName, YCurrent 238  
get\_logicalName, YDataLogger 276  
get\_logicalName, YDigitalIO 337  
get\_logicalName, YDisplay 387  
get\_logicalName, YDualPower 455  
get\_logicalName, YFiles 484  
get\_logicalName, YGenericSensor 513  
get\_logicalName, YGyro 560  
get\_logicalName, YHubPort 606  
get\_logicalName, YHumidity 635  
get\_logicalName, YLed 669  
get\_logicalName, YLightSensor 702  
get\_logicalName, YMagnetometer 741  
get\_logicalName, YModule 789  
get\_logicalName, YNetwork 838  
get\_logicalName, YOsControl 883  
get\_logicalName, YPower 913  
get\_logicalName, YPressure 955  
get\_logicalName, YPwmOutput 995  
get\_logicalName, YPwmPowerSource 1026  
get\_logicalName, YQt 1055  
get\_logicalName, YRealTimeClock 1089  
get\_logicalName, YRefFrame 1123  
get\_logicalName, YRelay 1154  
get\_logicalName, YSensor 1193  
get\_logicalName, YServo 1229  
get\_logicalName, YTemperature 1267  
get\_logicalName, YTilt 1308  
get\_logicalName, YVoc 1347  
get\_logicalName, YVoltage 1386  
get\_logicalName, YVSource 1421  
get\_logicalName, YWakeUpMonitor 1452  
get\_logicalName, YWakeUpSchedule 1488  
get\_logicalName, YWatchdog 1527  
get\_logicalName, YWireless 1572  
get\_lowestValue, YAccelerometer 51  
get\_lowestValue, YCarbonDioxide 131  
get\_lowestValue, YCompass 199  
get\_lowestValue, YCurrent 239  
get\_lowestValue, YGenericSensor 514  
get\_lowestValue, YGyro 561  
get\_lowestValue, YHumidity 636  
get\_lowestValue, YLightSensor 703  
get\_lowestValue, YMagnetometer 742  
get\_lowestValue, YPower 914  
get\_lowestValue, YPressure 956  
get\_lowestValue, YQt 1056  
get\_lowestValue, YSensor 1194  
get\_lowestValue, YTemperature 1268  
get\_lowestValue, YTilt 1309

get\_lowestValue, YVoc 1348  
get\_lowestValue, YVoltage 1387  
get\_luminosity, YLed 670  
get\_luminosity, YModule 790  
get\_macAddress, YNetwork 839  
get\_magneticHeading, YCompass 200  
get\_maxTimeOnStateA, YRelay 1155  
get\_maxTimeOnStateA, YWatchdog 1528  
get\_maxTimeOnStateB, YRelay 1156  
get\_maxTimeOnStateB, YWatchdog 1529  
get\_maxValue, YDataStream 313  
get\_maxValue, YMeasure 768  
get\_measures, YDataSet 297  
get\_message, YWireless 1573  
get\_meter, YPower 915  
get\_meterTimer, YPower 916  
get\_minutes, YWakeUpSchedule 1489  
get\_minutesA, YWakeUpSchedule 1490  
get\_minutesB, YWakeUpSchedule 1491  
get\_minValue, YDataStream 314  
get\_minValue, YMeasure 769  
get\_module, YAccelerometer 52  
get\_module, YAnButton 95  
get\_module, YCarbonDioxide 132  
get\_module, YColorLed 165  
get\_module, YCompass 201  
get\_module, YCurrent 240  
get\_module, YDataLogger 277  
get\_module, YDigitalIO 338  
get\_module, YDisplay 388  
get\_module, YDualPower 456  
get\_module, YFiles 485  
get\_module, YGenericSensor 515  
get\_module, YGyro 562  
get\_module, YHubPort 607  
get\_module, YHumidity 637  
get\_module, YLed 671  
get\_module, YLightSensor 704  
get\_module, YMagnetometer 743  
get\_module, YNetwork 840  
get\_module, YOsControl 884  
get\_module, YPower 917  
get\_module, YPressure 957  
get\_module, YPwmOutput 996  
get\_module, YPwmPowerSource 1027  
get\_module, YQt 1057  
get\_module, YRealTimeClock 1090  
get\_module, YRefFrame 1124  
get\_module, YRelay 1157  
get\_module, YSensor 1195  
get\_module, YServo 1230  
get\_module, YTemperature 1269  
get\_module, YTilt 1310  
get\_module, YVoc 1349  
get\_module, YVoltage 1388  
get\_module, YVSource 1422  
get\_module, YWakeUpMonitor 1453  
get\_module, YWakeUpSchedule 1492  
get\_module, YWatchdog 1530  
get\_module, YWireless 1574  
get\_monthDays, YWakeUpSchedule 1493  
get\_months, YWakeUpSchedule 1494  
get\_mountOrientation, YRefFrame 1125  
get\_mountPosition, YRefFrame 1126  
get\_neutral, YServo 1231  
get\_nextOccurrence, YWakeUpSchedule 1495  
get\_nextWakeUp, YWakeUpMonitor 1454  
get\_orientation, YDisplay 389  
get\_output, YRelay 1158  
get\_output, YWatchdog 1531  
get\_outputVoltage, YDigitalIO 339  
get\_overCurrent, YVSource 1423  
get\_overHeat, YVSource 1424  
get\_overLoad, YVSource 1425  
get\_period, YPwmOutput 997  
get\_persistentSettings, YModule 791  
get\_pitch, YGyro 563  
get\_poeCurrent, YNetwork 841  
get\_portDirection, YDigitalIO 340  
get\_portOpenDrain, YDigitalIO 341  
get\_portPolarity, YDigitalIO 342  
get\_portSize, YDigitalIO 343  
get\_portState, YDigitalIO 344  
get\_portState, YHubPort 608  
get\_position, YServo 1232  
get\_positionAtPowerOn, YServo 1233  
get\_power, YLed 672  
get\_powerControl, YDualPower 457  
get\_powerDuration, YWakeUpMonitor 1455  
get\_powerMode, YPwmPowerSource 1028  
get\_powerState, YDualPower 458  
get\_preview, YDataSet 298  
get\_primaryDNS, YNetwork 842  
get\_productId, YModule 792  
get\_productName, YModule 793  
get\_productRelease, YModule 794  
get\_progress, YDataSet 299  
get\_pulseCounter, YAnButton 96  
get\_pulseDuration, YPwmOutput 998  
get\_pulseTimer, YAnButton 97  
get\_pulseTimer, YRelay 1159  
get\_pulseTimer, YWatchdog 1532  
get\_quaternionW, YGyro 564  
get\_quaternionX, YGyro 565  
get\_quaternionY, YGyro 566  
get\_quaternionZ, YGyro 567  
get\_range, YServo 1234  
get\_rawValue, YAnButton 98  
get\_readiness, YNetwork 843  
get\_rebootCountdown, YModule 795  
get\_recordedData, YAccelerometer 53  
get\_recordedData, YCarbonDioxide 133  
get\_recordedData, YCompass 202  
get\_recordedData, YCurrent 241  
get\_recordedData, YGenericSensor 516  
get\_recordedData, YGyro 568  
get\_recordedData, YHumidity 638  
get\_recordedData, YLightSensor 705

get\_recordedData, YMagnetometer 744  
get\_recordedData, YPower 918  
get\_recordedData, YPressure 958  
get\_recordedData, YQt 1058  
get\_recordedData, YSensor 1196  
get\_recordedData, YTemperature 1270  
get\_recordedData, YTilt 1311  
get\_recordedData, YVoc 1350  
get\_recordedData, YVoltage 1389  
get\_recording, YDataLogger 278  
get\_regulationFailure, YVSource 1426  
get\_reportFrequency, YAccelerometer 54  
get\_reportFrequency, YCarbonDioxide 134  
get\_reportFrequency, YCompass 203  
get\_reportFrequency, YCurrent 242  
get\_reportFrequency, YGenericSensor 517  
get\_reportFrequency, YGyro 569  
get\_reportFrequency, YHumidity 639  
get\_reportFrequency, YLightSensor 706  
get\_reportFrequency, YMagnetometer 745  
get\_reportFrequency, YPower 919  
get\_reportFrequency, YPressure 959  
get\_reportFrequency, YQt 1059  
get\_reportFrequency, YSensor 1197  
get\_reportFrequency, YTemperature 1271  
get\_reportFrequency, YTilt 1312  
get\_reportFrequency, YVoc 1351  
get\_reportFrequency, YVoltage 1390  
get\_resolution, YAccelerometer 55  
get\_resolution, YCarbonDioxide 135  
get\_resolution, YCompass 204  
get\_resolution, YCurrent 243  
get\_resolution, YGenericSensor 518  
get\_resolution, YGyro 570  
get\_resolution, YHumidity 640  
get\_resolution, YLightSensor 707  
get\_resolution, YMagnetometer 746  
get\_resolution, YPower 920  
get\_resolution, YPressure 960  
get\_resolution, YQt 1060  
get\_resolution, YSensor 1198  
get\_resolution, YTemperature 1272  
get\_resolution, YTilt 1313  
get\_resolution, YVoc 1352  
get\_resolution, YVoltage 1391  
get\_rgbColor, YColorLed 166  
get\_rgbColorAtPowerOn, YColorLed 167  
get\_roll, YGyro 571  
get\_router, YNetwork 844  
getRowCount, YDataStream 315  
get\_runIndex, YDataStream 316  
get\_running, YWatchdog 1533  
get\_secondaryDNS, YNetwork 845  
get\_security, YWireless 1575  
get\_sensitivity, YAnButton 99  
get\_sensorType, YTemperature 1273  
get\_serialNumber, YModule 796  
get\_shutdownCountdown, YOsControl 885  
get\_signalRange, YGenericSensor 519  
get\_signalUnit, YGenericSensor 520  
get\_signalValue, YGenericSensor 521  
get\_sleepCountdown, YWakeUpMonitor 1456  
get\_ssId, YWireless 1576  
get\_startTime, YDataStream 317  
get\_startTimeUTC, YDataRun 291  
get\_startTimeUTC, YDataSet 300  
get\_startTimeUTC, YDataStream 318  
get\_startTimeUTC, YMeasure 770  
get\_startupSeq, YDisplay 390  
get\_state, YRelay 1160  
get\_state, YWatchdog 1534  
get\_stateAtPowerOn, YRelay 1161  
get\_stateAtPowerOn, YWatchdog 1535  
get\_subnetMask, YNetwork 846  
get\_summary, YDataSet 301  
get\_timeSet, YRealTimeClock 1091  
get\_timeUTC, YDataLogger 279  
get\_triggerDelay, YWatchdog 1536  
get\_triggerDuration, YWatchdog 1537  
get\_unit, YAccelerometer 56  
get\_unit, YCarbonDioxide 136  
get\_unit, YCompass 205  
get\_unit, YCurrent 244  
get\_unit, YDataSet 302  
get\_unit, YGenericSensor 522  
get\_unit, YGyro 572  
get\_unit, YHumidity 641  
get\_unit, YLightSensor 708  
get\_unit, YMagnetometer 747  
get\_unit, YPower 921  
get\_unit, YPressure 961  
get\_unit, YQt 1061  
get\_unit, YSensor 1199  
get\_unit, YTemperature 1274  
get\_unit, YTilt 1314  
get\_unit, YVoc 1353  
get\_unit, YVoltage 1392  
get\_unit, YVSource 1427  
get\_unixTime, YRealTimeClock 1092  
get\_upTime, YModule 797  
get\_usbBandwidth, YModule 798  
get\_usbCurrent, YModule 799  
get\_userData, YAccelerometer 57  
get\_userData, YAnButton 100  
get\_userData, YCarbonDioxide 137  
get\_userData, YColorLed 168  
get\_userData, YCompass 206  
get\_userData, YCurrent 245  
get\_userData, YDataLogger 280  
get\_userData, YDigitalIO 345  
get\_userData, YDisplay 391  
get\_userData, YDualPower 459  
get\_userData, YFiles 486  
get\_userData, YGenericSensor 523  
get\_userData, YGyro 573  
get\_userData, YHubPort 609  
get\_userData, YHumidity 642  
get\_userData, YLed 673

get(userData, YLightSensor 709  
get(userData, YMagnetometer 748  
get(userData, YModule 800  
get(userData, YNetwork 847  
get(userData, YOsControl 886  
get(userData, YPower 922  
get(userData, YPressure 962  
get(userData, YPwmOutput 999  
get(userData, YPwmPowerSource 1029  
get(userData, YQt 1062  
get(userData, YRealTimeClock 1093  
get(userData, YRefFrame 1127  
get(userData, YRelay 1162  
get(userData, YSensor 1200  
get(userData, YServo 1235  
get(userData, YTTemperature 1275  
get(userData, YTilt 1315  
get(userData, YVoc 1354  
get(userData, YVoltage 1393  
get(userData, YVSource 1428  
get(userData, YWakeUpMonitor 1457  
get(userData, YWakeUpSchedule 1496  
get(userData, YWatchdog 1538  
get(userData, YWireless 1577  
get(userPassword, YNetwork 848  
get\_utcOffset, YRealTimeClock 1094  
get\_valueRange, YGenericSensor 524  
get\_voltage, YVSource 1429  
get\_wakeUpReason, YWakeUpMonitor 1458  
get\_wakeUpState, YWakeUpMonitor 1459  
get\_weekDays, YWakeUpSchedule 1497  
get\_wwwWatchdogDelay, YNetwork 849  
get\_xValue, YAccelerometer 58  
get\_xValue, YGyro 574  
get\_xValue, YMagnetometer 749  
get\_yValue, YAccelerometer 59  
get\_yValue, YGyro 575  
get\_yValue, YMagnetometer 750  
get\_zValue, YAccelerometer 60  
get\_zValue, YGyro 576  
get\_zValue, YMagnetometer 751  
GetAPIVersion, YAPI 18  
GetTickCount, YAPI 19  
Gyro 542

1077, 1140, 1176, 1215, 1250, 1291, 1330,  
1369, 1408, 1440, 1475, 1512, 1557

Introduction 1

isOnline, YAccelerometer 61  
isOnline, YAnButton 101  
isOnline, YCarbonDioxide 138  
isOnline, YColorLed 170  
isOnline, YCompass 207  
isOnline, YCurrent 246  
isOnline, YDataLogger 281  
isOnline, YDigitalIO 346  
isOnline, YDisplay 392  
isOnline, YDualPower 460  
isOnline, YFiles 487  
isOnline, YGenericSensor 525  
isOnline, YGyro 577  
isOnline, YHubPort 610  
isOnline, YHumidity 643  
isOnline, YLed 674  
isOnline, YLightSensor 710  
isOnline, YMagnetometer 752  
isOnline, YModule 801  
isOnline, YNetwork 850  
isOnline, YOsControl 887  
isOnline, YPower 923  
isOnline, YPressure 963  
isOnline, YPwmOutput 1000  
isOnline, YPwmPowerSource 1030  
isOnline, YQt 1063  
isOnline, YRealTimeClock 1095  
isOnline, YRefFrame 1128  
isOnline, YRelay 1163  
isOnline, YSensor 1201  
isOnline, YServo 1236  
isOnline, YTTemperature 1276  
isOnline, YTilt 1316  
isOnline, YVoc 1355  
isOnline, YVoltage 1394  
isOnline, YVSource 1430  
isOnline, YWakeUpMonitor 1460  
isOnline, YWakeUpSchedule 1498  
isOnline, YWatchdog 1539  
isOnline, YWireless 1578

## H

HandleEvents, YAPI 20  
hide, YDisplayLayer 428  
Horloge 1077  
hslMove, YColorLed 169  
Humidity 618

## I

InitAPI, YAPI 21  
Intégration 8  
Interface 33, 75, 113, 152, 181, 221, 260, 320,  
364, 411, 443, 468, 496, 542, 593, 618, 657,  
684, 724, 772, 815, 895, 938, 977, 1015, 1038,

## J

joinNetwork, YWireless 1579

## L

Librairie 8  
LightSensor 684  
lineTo, YDisplayLayer 429  
load, YAccelerometer 62  
load, YAnButton 102  
load, YCarbonDioxide 139  
load, YColorLed 171  
load, YCompass 208  
load, YCurrent 247  
load, YDataLogger 282

load, YDigitalIO 347  
load, YDisplay 393  
load, YDualPower 461  
load, YFiles 488  
load, YGenericSensor 526  
load, YGyro 578  
load, YHubPort 611  
load, YHumidity 644  
load, YLed 675  
load, YLightSensor 711  
load, YMagnetometer 753  
load, YModule 802  
load, YNetwork 851  
load, YOsControl 888  
load, YPower 924  
load, YPressure 964  
load, YPwmOutput 1001  
load, YPwmPowerSource 1031  
load, YQt 1064  
load, YRealTimeClock 1096  
load, YRefFrame 1129  
load, YRelay 1164  
load, YSensor 1202  
load,YServo 1237  
load, YTemperature 1277  
load, YTilt 1317  
load, YVoc 1356  
load, YVoltage 1395  
load, YVSource 1431  
load, YWakeUpMonitor 1461  
load, YWakeUpSchedule 1499  
load, YWatchdog 1540  
load, YWireless 1580  
loadCalibrationPoints, YAccelerometer 63  
loadCalibrationPoints, YCarbonDioxide 140  
loadCalibrationPoints, YCompass 209  
loadCalibrationPoints, YCurrent 248  
loadCalibrationPoints, YGenericSensor 527  
loadCalibrationPoints, YGyro 579  
loadCalibrationPoints, YHumidity 645  
loadCalibrationPoints, YLightSensor 712  
loadCalibrationPoints, YMagnetometer 754  
loadCalibrationPoints, YPower 925  
loadCalibrationPoints, YPressure 965  
loadCalibrationPoints, YQt 1065  
loadCalibrationPoints, YSensor 1203  
loadCalibrationPoints, YTemperature 1278  
loadCalibrationPoints, YTilt 1318  
loadCalibrationPoints, YVoc 1357  
loadCalibrationPoints, YVoltage 1396  
loadMore, YDataSet 303

## M

Magnetometer 724  
Mesurée 766  
Mise 291  
Module 5, 772  
more3DCalibration, YRefFrame 1130  
move, YServo 1238

moveTo, YDisplayLayer 430

## N

Network 815  
newSequence, YDisplay 394  
nextAccelerometer, YAccelerometer 64  
nextAnButton, YAnButton 103  
nextCarbonDioxide, YCarbonDioxide 141  
nextColorLed, YColorLed 172  
nextCompass, YCompass 210  
nextCurrent, YCurrent 249  
nextDataLogger, YDataLogger 283  
nextDigitalIO, YDigitalIO 348  
nextDisplay, YDisplay 395  
nextDualPower, YDualPower 462  
nextFiles, YFiles 489  
nextGenericSensor, YGenericSensor 528  
nextGyro, YGyro 580  
nextHubPort, YHubPort 612  
nextHumidity, YHumidity 646  
nextLed, YLed 676  
nextLightSensor, YLightSensor 713  
nextMagnetometer, YMagnetometer 755  
nextModule, YModule 803  
nextNetwork, YNetwork 852  
nextOsControl, YOsControl 889  
nextPower, YPower 926  
nextPressure, YPressure 966  
nextPwmOutput, YPwmOutput 1002  
nextPwmPowerSource, YPwmPowerSource 1032  
nextQt, YQt 1066  
nextRealTimeClock, YRealTimeClock 1097  
nextRefFrame, YRefFrame 1131  
nextRelay, YRelay 1165  
nextSensor, YSensor 1204  
nextServo, YServo 1239  
nextTemperature, YTemperature 1279  
nextTilt, YTilt 1319  
nextVoc, YVoc 1358  
nextVoltage, YVoltage 1397  
nextVSource, YVSource 1432  
nextWakeUpMonitor, YWakeUpMonitor 1462  
nextWakeUpSchedule, YWakeUpSchedule 1500  
nextWatchdog, YWatchdog 1541  
nextWireless, YWireless 1581

## O

Objets 411

## P

pauseSequence, YDisplay 396  
ping, YNetwork 853  
playSequence, YDisplay 397  
Port 593  
Power 895  
PreregisterHub, YAPI 22

Pressure 938  
pulse, YDigitalIO 349  
pulse, YRelay 1166  
pulse, YVSource 1433  
pulse, YWatchdog 1542  
pulseDurationMove, YPwmOutput 1003  
PwmPowerSource 1015

## Q

Quaternion 1038

## R

Real 1077  
reboot, YModule 804  
Reference 12  
Référentiel 1104  
registerAnglesCallback, YGyro 581  
RegisterDeviceArrivalCallback, YAPI 23  
RegisterDeviceRemovalCallback, YAPI 24  
RegisterHub, YAPI 25  
RegisterHubDiscoveryCallback, YAPI 26  
registerLogCallback, YModule 805  
RegisterLogFunction, YAPI 27  
registerQuaternionCallback, YGyro 582  
registerTimedReportCallback, YAccelerometer 65  
registerTimedReportCallback, YCarbonDioxide 142  
registerTimedReportCallback, YCompass 211  
registerTimedReportCallback, YCurrent 250  
registerTimedReportCallback, YGenericSensor 529  
registerTimedReportCallback, YGyro 583  
registerTimedReportCallback, YHumidity 647  
registerTimedReportCallback, YLightSensor 714  
registerTimedReportCallback, YMagnetometer 756  
registerTimedReportCallback, YPower 927  
registerTimedReportCallback, YPressure 967  
registerTimedReportCallback, YQt 1067  
registerTimedReportCallback,YSensor 1205  
registerTimedReportCallback, YTemperature 1280  
registerTimedReportCallback, YTilt 1320  
registerTimedReportCallback, YVoc 1359  
registerTimedReportCallback, YVoltage 1398  
registerValueCallback, YAccelerometer 66  
registerValueCallback, YAnButton 104  
registerValueCallback, YCarbonDioxide 143  
registerValueCallback, YColorLed 173  
registerValueCallback, YCompass 212  
registerValueCallback, YCurrent 251  
registerValueCallback, YDataLogger 284  
registerValueCallback, YDigitalIO 350  
registerValueCallback, YDisplay 398  
registerValueCallback, YDualPower 463  
registerValueCallback, YFiles 490  
registerValueCallback, YGenericSensor 530

registerValueCallback, YGyro 584  
registerValueCallback, YHubPort 613  
registerValueCallback, YHumidity 648  
registerValueCallback, YLed 677  
registerValueCallback, YLightSensor 715  
registerValueCallback, YMagnetometer 757  
registerValueCallback, YNetwork 854  
registerValueCallback, YOsControl 890  
registerValueCallback, YPower 928  
registerValueCallback, YPressure 968  
registerValueCallback, YPwmOutput 1004  
registerValueCallback, YPwmPowerSource 1033  
registerValueCallback, YQt 1068  
registerValueCallback, YRealTimeClock 1098  
registerValueCallback, YRefFrame 1132  
registerValueCallback, YRelay 1167  
registerValueCallback, YSensor 1206  
registerValueCallback, YServo 1240  
registerValueCallback, YTemperature 1281  
registerValueCallback, YTilt 1321  
registerValueCallback, YVoc 1360  
registerValueCallback, YVoltage 1399  
registerValueCallback, YVSource 1434  
registerValueCallback, YWakeUpMonitor 1463  
registerValueCallback, YWakeUpSchedule 1501  
registerValueCallback, YWatchdog 1543  
registerValueCallback, YWireless 1582  
Relay 1140  
remove, YFiles 491  
reset, YDisplayLayer 431  
reset, YPower 929  
resetAll, YDisplay 399  
resetCounter, YAnButton 105  
resetSleepCountDown, YWakeUpMonitor 1464  
resetWatchdog, YWatchdog 1544  
revertFromFlash, YModule 806  
rgbMove, YColorLed 174

## S

save3DCalibration, YRefFrame 1133  
saveSequence, YDisplay 400  
saveToFlash, YModule 807  
selectColorPen, YDisplayLayer 432  
selectEraser, YDisplayLayer 433  
selectFont, YDisplayLayer 434  
selectGrayPen, YDisplayLayer 435  
Senseur 1176  
Séquence 291, 293, 305  
Servo 1215  
set\_adminPassword, YNetwork 855  
set\_analogCalibration, YAnButton 106  
set\_autoStart, YDataLogger 285  
set\_autoStart, YWatchdog 1545  
set\_beacon, YModule 808  
set\_bearing, YRefFrame 1134  
set\_bitDirection, YDigitalIO 351  
set\_bitOpenDrain, YDigitalIO 352  
set\_bitPolarity, YDigitalIO 353  
set\_bitState, YDigitalIO 354

set\_blinking, YLed 678  
set\_brightness, YDisplay 401  
set\_calibrationMax, YAnButton 107  
set\_calibrationMin, YAnButton 108  
set\_callbackCredentials, YNetwork 856  
set\_callbackEncoding, YNetwork 857  
set\_callbackMaxDelay, YNetwork 858  
set\_callbackMethod, YNetwork 859  
set\_callbackMinDelay, YNetwork 860  
set\_callbackUrl, YNetwork 861  
set\_discoverable, YNetwork 862  
set\_dutyCycle, YPwmOutput 1005  
set\_dutyCycleAtPowerOn, YPwmOutput 1006  
set\_enabled, YDisplay 402  
set\_enabled, YHubPort 614  
set\_enabled, YPwmOutput 1007  
set\_enabled,YServo 1241  
set\_enabledAtPowerOn, YPwmOutput 1008  
set\_enabledAtPowerOn, YServo 1242  
set\_frequency, YPwmOutput 1009  
set\_highestValue, YAccelerometer 67  
set\_highestValue, YCarbonDioxide 144  
set\_highestValue, YCompass 213  
set\_highestValue, YCurrent 252  
set\_highestValue, YGenericSensor 531  
set\_highestValue, YGyro 585  
set\_highestValue, YHumidity 649  
set\_highestValue, YLightSensor 716  
set\_highestValue, YMagnetometer 758  
set\_highestValue, YPower 930  
set\_highestValue, YPressure 969  
set\_highestValue, YQt 1069  
set\_highestValue, YSensor 1207  
set\_highestValue, YTemperature 1282  
set\_highestValue, YTilt 1322  
set\_highestValue, YVoc 1361  
set\_highestValue, YVoltage 1400  
set\_hours, YWakeUpSchedule 1502  
set\_hslColor, YColorLed 175  
set\_logFrequency, YAccelerometer 68  
set\_logFrequency, YCarbonDioxide 145  
set\_logFrequency, YCompass 214  
set\_logFrequency, YCurrent 253  
set\_logFrequency, YGenericSensor 532  
set\_logFrequency, YGyro 586  
set\_logFrequency, YHumidity 650  
set\_logFrequency, YLightSensor 717  
set\_logFrequency, YMagnetometer 759  
set\_logFrequency, YPower 931  
set\_logFrequency, YPressure 970  
set\_logFrequency, YQt 1070  
set\_logFrequency, YSensor 1208  
set\_logFrequency, YTemperature 1283  
set\_logFrequency, YTilt 1323  
set\_logFrequency, YVoc 1362  
set\_logFrequency, YVoltage 1401  
set\_logicalName, YAccelerometer 69  
set\_logicalName, YAnButton 109  
set\_logicalName, YCarbonDioxide 146  
set\_logicalName, YColorLed 176  
set\_logicalName, YCompass 215  
set\_logicalName, YCurrent 254  
set\_logicalName, YDataLogger 286  
set\_logicalName, YDigitalIO 355  
set\_logicalName, YDisplay 403  
set\_logicalName, YDualPower 464  
set\_logicalName, YFiles 492  
set\_logicalName, YGenericSensor 533  
set\_logicalName, YGyro 587  
set\_logicalName, YHubPort 615  
set\_logicalName, YHumidity 651  
set\_logicalName, YLed 679  
set\_logicalName, YLightSensor 718  
set\_logicalName, YMagnetometer 760  
set\_logicalName, YModule 809  
set\_logicalName, YNetwork 863  
set\_logicalName, YOsControl 891  
set\_logicalName, YPower 932  
set\_logicalName, YPressure 971  
set\_logicalName, YPwmOutput 1010  
set\_logicalName, YPwmPowerSource 1034  
set\_logicalName, YQt 1071  
set\_logicalName, YRealTimeClock 1099  
set\_logicalName, YRefFrame 1135  
set\_logicalName, YRelay 1168  
set\_logicalName, YSensor 1209  
set\_logicalName, YServo 1243  
set\_logicalName, YTemperature 1284  
set\_logicalName, YTilt 1324  
set\_logicalName, YVoc 1363  
set\_logicalName, YVoltage 1402  
set\_logicalName, YVSource 1435  
set\_logicalName, YWakeUpMonitor 1465  
set\_logicalName, YWakeUpSchedule 1503  
set\_logicalName, YWatchdog 1546  
set\_logicalName, YWireless 1583  
set\_lowestValue, YAccelerometer 70  
set\_lowestValue, YCarbonDioxide 147  
set\_lowestValue, YCompass 216  
set\_lowestValue, YCurrent 255  
set\_lowestValue, YGenericSensor 534  
set\_lowestValue, YGyro 588  
set\_lowestValue, YHumidity 652  
set\_lowestValue, YLightSensor 719  
set\_lowestValue, YMagnetometer 761  
set\_lowestValue, YPower 933  
set\_lowestValue, YPressure 972  
set\_lowestValue, YQt 1072  
set\_lowestValue, YSensor 1210  
set\_lowestValue, YTemperature 1285  
set\_lowestValue, YTilt 1325  
set\_lowestValue, YVoc 1364  
set\_lowestValue, YVoltage 1403  
set\_luminosity, YLed 680  
set\_luminosity, YModule 810  
set\_maxTimeOnStateA, YRelay 1169  
set\_maxTimeOnStateA, YWatchdog 1547  
set\_maxTimeOnStateB, YRelay 1170

set\_maxTimeOnStateB, YWatchdog 1548  
set\_minutes, YWakeUpSchedule 1504  
set\_minutesA, YWakeUpSchedule 1505  
set\_minutesB, YWakeUpSchedule 1506  
set\_monthDays, YWakeUpSchedule 1507  
set\_months, YWakeUpSchedule 1508  
set\_mountPosition, YRefFrame 1136  
set\_neutral,YServo 1244  
set\_nextWakeUp, YWakeUpMonitor 1466  
set\_orientation, YDisplay 404  
set\_output, YRelay 1171  
set\_output, YWatchdog 1549  
set\_outputVoltage, YDigitalIO 356  
set\_period, YPwmOutput 1011  
set\_portDirection, YDigitalIO 357  
set\_portOpenDrain, YDigitalIO 358  
set\_portPolarity, YDigitalIO 359  
set\_portState, YDigitalIO 360  
set\_position, YServo 1245  
set\_positionAtPowerOn, YServo 1246  
set\_power, YLed 681  
set\_powerControl, YDualPower 465  
set\_powerDuration, YWakeUpMonitor 1467  
set\_powerMode, YPwmPowerSource 1035  
set\_primaryDNS, YNetwork 864  
set\_pulseDuration, YPwmOutput 1012  
set\_range, YServo 1247  
set\_recording, YDataLogger 287  
set\_reportFrequency, YAccelerometer 71  
set\_reportFrequency, YCarbonDioxide 148  
set\_reportFrequency, YCompass 217  
set\_reportFrequency, YCurrent 256  
set\_reportFrequency, YGenericSensor 535  
set\_reportFrequency, YGyro 589  
set\_reportFrequency, YHumidity 653  
set\_reportFrequency, YLightSensor 720  
set\_reportFrequency, YMagnetometer 762  
set\_reportFrequency, YPower 934  
set\_reportFrequency, YPressure 973  
set\_reportFrequency, YQt 1073  
set\_reportFrequency, YSensor 1211  
set\_reportFrequency, YTemperature 1286  
set\_reportFrequency, YTilt 1326  
set\_reportFrequency, YVoc 1365  
set\_reportFrequency, YVoltage 1404  
set\_resolution, YAccelerometer 72  
set\_resolution, YCarbonDioxide 149  
set\_resolution, YCompass 218  
set\_resolution, YCurrent 257  
set\_resolution, YGenericSensor 536  
set\_resolution, YGyro 590  
set\_resolution, YHumidity 654  
set\_resolution, YLightSensor 721  
set\_resolution, YMagnetometer 763  
set\_resolution, YPower 935  
set\_resolution, YPressure 974  
set\_resolution, YQt 1074  
set\_resolution, YSensor 1212  
set\_resolution, YTemperature 1287  
set\_resolution, YTilt 1327  
set\_resolution, YVoc 1366  
set\_resolution, YVoltage 1405  
set\_rgbColor, YColorLed 177  
set\_rgbColorAtPowerOn, YColorLed 178  
set\_running, YWatchdog 1550  
set\_secondaryDNS, YNetwork 865  
set\_sensitivity, YAnButton 110  
set\_sensorType, YTemperature 1288  
set\_signalRange, YGenericSensor 537  
set\_sleepCountdown, YWakeUpMonitor 1468  
set\_startupSeq, YDisplay 405  
set\_state, YRelay 1172  
set\_state, YWatchdog 1551  
set\_stateAtPowerOn, YRelay 1173  
set\_stateAtPowerOn, YWatchdog 1552  
set\_timeUTC, YDataLogger 288  
set\_triggerDelay, YWatchdog 1553  
set\_triggerDuration, YWatchdog 1554  
set\_unit, YGenericSensor 538  
set\_unixTime, YRealTimeClock 1100  
set\_usbBandwidth, YModule 811  
set\_userData, YAccelerometer 73  
set\_userData, YAnButton 111  
set\_userData, YCarbonDioxide 150  
set\_userData, YColorLed 179  
set\_userData, YCompass 219  
set\_userData, YCurrent 258  
set\_userData, YDataLogger 289  
set\_userData, YDigitalIO 361  
set\_userData, YDisplay 406  
set\_userData, YDualPower 466  
set\_userData, YFiles 493  
set\_userData, YGenericSensor 539  
set\_userData, YGyro 591  
set\_userData, YHubPort 616  
set\_userData, YHumidity 655  
set\_userData, YLed 682  
set\_userData, YLightSensor 722  
set\_userData, YMagnetometer 764  
set\_userData, YModule 812  
set\_userData, YNetwork 866  
set\_userData, YOsControl 892  
set\_userData, YPower 936  
set\_userData, YPressure 975  
set\_userData, YPwmOutput 1013  
set\_userData, YPwmPowerSource 1036  
set\_userData, YQt 1075  
set\_userData, YRealTimeClock 1101  
set\_userData, YRefFrame 1137  
set\_userData, YRelay 1174  
set\_userData, YSensor 1213  
set\_userData, YServo 1248  
set\_userData, YTemperature 1289  
set\_userData, YTilt 1328  
set\_userData, YVoc 1367  
set\_userData, YVoltage 1406  
set\_userData, YVSource 1436  
set\_userData, YWakeUpMonitor 1469

set(userData, YWakeUpSchedule) 1509  
set(userData, YWatchdog) 1555  
set(userData, YWireless) 1584  
set(userPassword, YNetwork) 867  
set\_utcOffset, YRealTimeClock 1102  
set\_valueRange, YGenericSensor 540  
set\_voltage, YVSource 1437  
set\_weekDays, YWakeUpSchedule 1510  
set\_wwwWatchdogDelay, YNetwork 868  
setAntialiasingMode, YDisplayLayer 436  
setConsoleBackground, YDisplayLayer 437  
setConsoleMargins, YDisplayLayer 438  
setConsoleWordWrap, YDisplayLayer 439  
setLayerPosition, YDisplayLayer 440  
shutdown, YOsControl 893  
Sleep, YAPI 28  
sleep, YWakeUpMonitor 1470  
sleepFor, YWakeUpMonitor 1471  
sleepUntil, YWakeUpMonitor 1472  
Source 1408  
start3DCalibration, YRefFrame 1138  
stopSequence, YDisplay 407  
swapLayerContent, YDisplay 408

## T

Temperature 1250  
Temps 1077  
Tension 1408  
Tilt 1291  
toggle\_bitState, YDigitalIO 362  
triggerFirmwareUpdate, YModule 813  
TriggerHubDiscovery, YAPI 29  
Type 1176

## U

unhide, YDisplayLayer 441  
UnregisterHub, YAPI 30  
UpdateDeviceList, YAPI 31  
upload, YDisplay 409  
upload, YFiles 494  
useDHCP, YNetwork 869  
useStaticIP, YNetwork 870

## V

Valeur 766  
Voltage 1369  
voltageMove, YVSource 1438

## W

wakeUp, YWakeUpMonitor 1473  
WakeUpMonitor 1440  
WakeUpSchedule 1475  
Watchdog 1512  
Wireless 1557

## Y

YAccelerometer 35-73  
YAnButton 77-111  
YAPI 14-31  
YCarbonDioxide 115-150  
yCheckLogicalName 14  
YColorLed 153-179  
YCompass 183-219  
YCurrent 223-258  
YDataLogger 261-289  
YDataRun 291  
YDataSet 294-303  
YDataStream 306-318  
YDigitalIO 322-362  
yDisableExceptions 15  
YDisplay 366-409  
YDisplayLayer 412-441  
YDualPower 444-466  
yEnableExceptions 16  
YFiles 469-494  
yFindAccelerometer 35  
yFindAnButton 77  
yFindCarbonDioxide 115  
yFindColorLed 153  
yFindCompass 183  
yFindCurrent 223  
yFindDataLogger 261  
yFindDigitalIO 322  
yFindDisplay 366  
yFindDualPower 444  
yFindFiles 469  
yFindGenericSensor 498  
yFindGyro 544  
yFindHubPort 594  
yFindHumidity 620  
yFindLed 658  
yFindLightSensor 686  
yFindMagnetometer 726  
yFindModule 774  
yFindNetwork 818  
yFindOsControl 873  
yFindPower 897  
yFindPressure 940  
yFindPwmOutput 979  
yFindPwmPowerSource 1016  
yFindQt 1040  
yFindRealTimeClock 1078  
yFindRefFrame 1106  
yFindRelay 1142  
yFindSensor 1178  
yFindServo 1217  
yFindTemperature 1252  
yFindTilt 1293  
yFindVoc 1332  
yFindVoltage 1371  
yFindVSource 1409  
yFindWakeUpMonitor 1442  
yFindWakeUpSchedule 1477

yFindWatchdog 1514  
yFindWireless 1558  
yFirstAccelerometer 36  
yFirstAnButton 78  
yFirstCarbonDioxide 116  
yFirstColorLed 154  
yFirstCompass 184  
yFirstCurrent 224  
yFirstDataLogger 262  
yFirstDigitalIO 323  
yFirstDisplay 367  
yFirstDualPower 445  
yFirstFiles 470  
yFirstGenericSensor 499  
yFirstGyro 545  
yFirstHubPort 595  
yFirstHumidity 621  
yFirstLed 659  
yFirstLightSensor 687  
yFirstMagnetometer 727  
yFirstModule 775  
yFirstNetwork 819  
yFirstOsControl 874  
yFirstPower 898  
yFirstPressure 941  
yFirstPwmOutput 980  
yFirstPwmPowerSource 1017  
yFirstQt 1041  
yFirstRealTimeClock 1079  
yFirstRefFrame 1107  
yFirstRelay 1143  
yFirstSensor 1179  
yFirstServo 1218  
yFirstTemperature 1253  
yFirstTilt 1294  
yFirstVoc 1333  
yFirstVoltage 1372  
yFirstVSource 1410  
yFirstWakeUpMonitor 1443  
yFirstWakeUpSchedule 1478  
yFirstWatchdog 1515  
yFirstWireless 1559  
yFreeAPI 17  
YGenericSensor 498-540  
yGetAPIVersion 18  
yGetTickCount 19  
YGyro 544-591  
yHandleEvents 20  
YHubPort 594-616  
YHumidity 620-655  
yInitAPI 21  
YLed 658-682  
YLightSensor 686-722  
YMagnetometer 726-764  
YMeasure 766-770  
YModule 774-813  
YNetwork 818-870  
Yocto-Demo 3  
Yocto-hub 593  
YOsControl 873-893  
YPower 897-936  
yPreregisterHub 22  
YPressure 940-975  
YPwmOutput 979-1013  
YPwmPowerSource 1016-1036  
YQt 1040-1075  
YRealTimeClock 1078-1102  
YRefFrame 1106-1138  
yRegisterDeviceArrivalCallback 23  
yRegisterDeviceRemovalCallback 24  
yRegisterHub 25  
yRegisterHubDiscoveryCallback 26  
yRegisterLogFunction 27  
YRelay 1142-1174  
YSensor 1178-1213  
YServo 1217-1248  
ySleep 28  
YTemperature 1252-1289  
YTilt 1293-1328  
yTriggerHubDiscovery 29  
yUnregisterHub 30  
yUpdateDeviceList 31  
YVoc 1332-1367  
YVoltage 1371-1406  
YVSource 1409-1438  
YWakeUpMonitor 1442-1473  
YWakeUpSchedule 1477-1510  
YWatchdog 1514-1555  
YWireless 1558-1584