



## Référence de l'API JavaScript



# Table des matières

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Utilisation du Yocto-Demo en Javascript .....</b>	<b>3</b>
2.1. Préparation .....	3
2.2. Contrôle de la fonction Led .....	3
2.3. Contrôle de la partie module .....	5
2.4. Gestion des erreurs .....	8
Blueprint .....	12
<b>3. Reference .....</b>	<b>12</b>
3.1. Fonctions générales .....	13
3.2. Interface de la fonction Accelerometer .....	32
3.3. Interface de la fonction AnButton .....	78
3.4. Interface de la fonction CarbonDioxide .....	120
3.5. Interface de la fonction ColorLed .....	163
3.6. Interface de la fonction Compass .....	196
3.7. Interface de la fonction Current .....	240
3.8. Interface de la fonction DataLogger .....	283
3.9. Séquence de données mise en forme .....	318
3.10. Séquence de données enregistrées .....	328
3.11. Séquence de données enregistrées brute .....	341
3.12. Interface de la fonction DigitalIO .....	356
3.13. Interface de la fonction Display .....	404
3.14. Interface des objets DisplayLayer .....	455
3.15. Interface de contrôle de l'alimentation .....	487
3.16. Interface de la fonction Files .....	516
3.17. Interface de la fonction GenericSensor .....	549
3.18. Interface de la fonction Gyro .....	599
3.19. Interface d'un port de Yocto-hub .....	654
3.20. Interface de la fonction Humidity .....	683
3.21. Interface de la fonction Led .....	726
3.22. Interface de la fonction LightSensor .....	757
3.23. Interface de la fonction Magnetometer .....	801
3.24. Valeur mesurée .....	847
3.25. Interface de contrôle du module .....	853

3.26. Interface de la fonction Network .....	898
3.27. contrôle d'OS .....	959
3.28. Interface de la fonction Power .....	986
3.29. Interface de la fonction Pressure .....	1033
3.30. Interface de la fonction Pwm .....	1076
3.31. Interface de la fonction PwmPowerSource .....	1118
3.32. Interface du quaternion .....	1145
3.33. Interface de la fonction Horloge Temps Real .....	1188
3.34. Configuration du référentiel .....	1219
3.35. Interface de la fonction Relay .....	1259
3.36. Interface des fonctions de type senseur .....	1299
3.37. Interface de la fonction Servo .....	1342
3.38. Interface de la fonction Temperature .....	1381
3.39. Interface de la fonction Tilt .....	1426
3.40. Interface de la fonction Voc .....	1469
3.41. Interface de la fonction Voltage .....	1512
3.42. Interface de la fonction Source de tension .....	1555
3.43. Interface de la fonction WakeUpMonitor .....	1591
3.44. Interface de la fonction WakeUpSchedule .....	1630
3.45. Interface de la fonction Watchdog .....	1671
3.46. Interface de la fonction Wireless .....	1720
<b>Index .....</b>	<b>1753</b>

# 1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie JavaScript de Yoctopuce pour interfaçer 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 Javascript

Javascript n'est probablement pas le premier langage qui vous serait venu à l'esprit pour contrôler du matériel, mais il présente l'immense avantage d'être très facile à mettre en oeuvre: avec Javascript, il ne vous faudra qu'un éditeur de texte et un browser internet pour réaliser vos premiers essais.

Au moment de l'écriture de ce manuel, la librairie Javascript fonctionne avec n'importe quel browser récent... sauf Opera. Il est probable que qu'Opera finira un jour par fonctionner avec la librairie Yoctopuce<sup>1</sup>, mais pour l'instant ce n'est pas le cas.

Javascript fait partie de ces langages qui ne vous permettront pas d'accéder directement aux couches matérielles de votre ordinateur. C'est pourquoi vous devrez faire tourner la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules

### 2.1. Préparation

Connectez vous sur le site de Yoctopuce et téléchargez les éléments suivants:

- La librairie de programmation pour Javascript<sup>2</sup>
- Le programme VirtualHub<sup>3</sup> pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

Décomprimez les fichiers de la librairie dans un répertoire de votre choix, branchez vos modules, lancez le programme VirtualHub,et vous pouvez commencer vos premiers test. Vous n'avez pas besoin d'installer de driver.

### 2.2. 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 JavaScript qui utilise la fonction Led.

```
<SCRIPT type="text/javascript" src="yocto_api.js">;</SCRIPT>
<SCRIPT type="text/javascript" src="yocto_led.js"></SCRIPT>

// On récupère l'objet représentant le module, à travers le VirtualHub local
yRegisterHub('http://127.0.0.1:4444/');
var led = yFindLed("YCTOPOC1-123456.led");

// Pour gérer le hot-plug, on vérifie que le module est là
```

<sup>1</sup> En fait dès qu'Opera implémentera le support pour le header HTTP Access-Control-Allow-Origin

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

<sup>3</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

```
if(led.isOnline())
{
    // Utiliser led.set_power(), ...
}
```

Voyons maintenant en détail ce que font ces quelques lignes.

## yocto\_api.js et yocto\_led.js

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

### yRegisterHub

La fonction `yRegisterHub` permet d'indiquer sur quelle machine se trouve les modules Yoctopuce, ou plus exactement la machine sur laquelle tourne le programme VirtualHub. Dans notre cas l'adresse `127.0.0.1:4444` indique la machine locale, en utilisant le port `4444` (le port standard utilisé par Yoctopuce). Vous pouvez parfaitement changer cette adresse, et mettre l'adresse d'une autre machine sur laquelle tournerait un autre VirtualHub.

### 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):

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

Ouvrez votre éditeur de texte préféré<sup>4</sup>, recopiez le code ci-dessous, sauvez-le dans le même répertoire que les fichiers de la librairie, et ouvrez-le avec votre browser favori (sauf Opera). Vous trouverez aussi ce code dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce.

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

L'exemple est codé pour être utilisé soit depuis un serveur web, soit en ouvrant directement le fichier localement sur la machine. Notez que cette dernière solution n'est pas possible avec certaines

<sup>4</sup> Si vous n'avez pas d'éditeur de texte, utilisez Notepad plutôt que Microsoft Word.

versions de Internet Explorer (en particulier IE 9 de Windows 7), qui refuse d'ouvrir des connexions réseau lorsqu'il travaille sur un fichier local. Pour utiliser Internet Explorer, vous devez donc mettre les pages sur un serveur web. Aucun problème par contre avec Chrome, Firefox ou Safari.

Si le Yocto-Demo n'est pas branché sur la machine où fonctionne le navigateur internet, remplacez dans l'exemple l'adresse 127.0.0.1 par l'adresse IP de la machine où est branché le Yocto-Demo et où vous avez lancé le VirtualHub.

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
<SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
<SCRIPT type="text/javascript" src="yocto_led.js"></SCRIPT>
<SCRIPT language='javascript1.5' type='text/JavaScript'>
<!--
// Use explicit error handling rather than exceptions
yDisableExceptions();

// Setup the API to use the VirtualHub on local machine
if(yRegisterHub('http://127.0.0.1:4444/') != YAPI_SUCCESS) {
    alert("Cannot contact VirtualHub on 127.0.0.1");
}

var led;

function refresh()
{
    var serial = document.getElementById('serial').value;
    if(serial == '') {
        // Detect any connected module suitable for the demo
        led = yFirstLed();
        if(led) {
            serial = led.module().get_serialNumber();
            document.getElementById('serial').value = serial;
        }
    }

    led = yFindLed(serial+".led");
    if(led.isOnline()) {
        document.getElementById('msg').value = '';
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout('refresh()',500);
}

function switchIt(state)
{
    if (state) led.set_power(Y_POWER_ON);
    else led.set_power(Y_POWER_OFF);
}
-->
</SCRIPT>
</HEAD>
<BODY onload='refresh();'>
Module to use: <input id='serial'>
<input id='msg' style='color:red;border:none;' readonly><br>
<a href='javascript:switchIt(true);'>ON</a><br>
<a href='javascript:switchIt(false);'>OFF</a>
</BODY>
</HTML>
```

## 2.3. 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.

```
<HTML>
<HEAD>
<TITLE>Module Control</TITLE>
```

```

<SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
<SCRIPT language='javascript1.5' type='text/JavaScript'>
<!--
// Use explicit error handling rather than exceptions
yDisableExceptions();

// Setup the API to use the VirtualHub on local machine
if(yRegisterHub('http://127.0.0.1:4444/') != YAPI_SUCCESS) {
    alert("Cannot contact VirtualHub on 127.0.0.1");
}

var module;

function refresh()
{
    var serial = document.getElementById('serial').value;
    if(serial == '') {
        // Detect any connected module suitable for the demo
        module = yFirstModule().nextModule();
        if(module) {
            serial = module.get_serialNumber();
            document.getElementById('serial').value = serial;
        }
    }

    module = yFindModule(serial);
    if(module.isOnline()) {
        document.getElementById('msg').value = '';
        var html = 'serial: '+module.get_serialNumber()+'<br>';
        html += 'logical name: '+module.get_logicalName()+'<br>';
        html += 'luminosity: '+module.get_luminosity()+'%<br>';
        html += 'beacon:';
        if (module.get_beacon() == Y_BEACON_ON)
            html+="ON <a href='javascript:beacon(Y_BEACON_OFF)'>switch off</a><br>";
        else
            html+="OFF <a href='javascript:beacon(Y_BEACON_ON)'>switch on</a><br>";
        html += 'upTime: '+parseInt(module.get_upTime()/1000)+' sec<br>';
        html += 'USB current: '+module.get_usbCurrent()+' mA<br>';
        html += 'logs:<br><pre>'+module.get_lastLogs()+'</pre><br>';
        document.getElementById('data').innerHTML = html;
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout('refresh()',1000);
}

function beacon(state)
{
    module.set_beacon(state);
    refresh();
}
-->
</SCRIPT>
</HEAD>
<BODY onload='refresh();'>
Module to use: <input id='serial'>
<input id='msg' style='color:red; border:none;' readonly><br>
<span id='data'></span>
</BODY>
</HTML>

```

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 chapitres 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.

```

<HTML>
<HEAD>
<TITLE>Change module settings</TITLE>
<SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
<SCRIPT language='javascript1.5' type='text/JavaScript'>
<!--
// Use explicit error handling rather than exceptions
yDisableExceptions();

// Setup the API to use the VirtualHub on local machine
if(yRegisterHub('http://127.0.0.1:4444/') != YAPI_SUCCESS) {
    alert("Cannot contact VirtualHub on 127.0.0.1");
}

var module;

function refresh()
{
    var serial = document.getElementById('serial').value;
    if(serial == '') {
        // Detect any connected module suitable for the demo
        module = yFirstModule().nextModule();
        if(module) {
            serial = module.get_serialNumber();
            document.getElementById('serial').value = serial;
        }
    }

    module = yFindModule(serial);
    if(module.isOnline()) {
        document.getElementById('msg').value = '';
        document.getElementById('curName').value = module.get_logicalName();
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout('refresh()',1000);
}

function save()
{
    var newname = document.getElementById('newName').value;
    if (!yCheckLogicalName(newname)) {
        alert('invalid logical name');
        return;
    }
    module.set_logicalName(newname);
    module.saveToFlash();
}
-->
</SCRIPT>
</HEAD>
<BODY onload='refresh();'>
Module to use: <input id='serial'>
<input id='msg' style='color:red;border:none;' readonly><br>
Current name: <input id='curName' readonly><br>
New logical name: <input id='newName'>
<a href='javascript:save();'>Save</a>
</BODY>
</HTML>

```

Attention, le nombre de cycle d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit de que la sauvegarde des réglages se passera correctement. Cette limite, lié à la technologie employé 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.

## Énumération 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

```
<HTML>
<HEAD>
<TITLE>Modules inventory</TITLE>
<SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
<SCRIPT language='javascript1.5' type='text/JavaScript'>
<!--
// Use explicit error handling rather than exceptions
yDisableExceptions();

// Setup the API to use the VirtualHub on local machine
if(yRegisterHub('http://127.0.0.1:4444/') != YAPI_SUCCESS) {
    alert("Cannot contact VirtualHub on 127.0.0.1");
}

function refresh()
{
    yUpdateDeviceList();

    var htmlcode = '';
    var module = yFirstModule();
    while(module) {
        htmlcode += module.get_serialNumber()
                    +'('+module.get_productName() +")<br>";
        module = module.nextModule();
    }
    document.getElementById('list').innerHTML=htmlcode;
    setTimeout('refresh()',500);
}
-->
</SCRIPT>
</HEAD>
<BODY onload='refresh();'>
<H1>Device list</H1>
<tt><span id='list'></span></tt>
</BODY>
</HTML>
```

## 2.4. 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 renournée 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 renournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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





### **3. Reference**

## 3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <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.

```
function yCheckLogicalName( 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.

```
function 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.

```
function 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.

```
function 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.

```
function 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).

```
function 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() yHandleEvents()yHandleEvents()

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

```
function yHandleEvents( 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.

```
function yInitAPI( mode, 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() yPreregisterHub(yPreregisterHub())

YAPI

Alternative plus tolérante à RegisterHub().

```
function yPreregisterHub( url, 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()****YAPI****yRegisterDeviceArrivalCallback()****yRegisterDeviceArrivalCallback()**

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

```
function yRegisterDeviceArrivalCallback( 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.

```
function yRegisterDeviceRemovalCallback( 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() yRegisterHub()yRegisterHub()

YAPI

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

```
function yRegisterHub( url, 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.SetTimeout() ySetTimeout()ySetTimeout()

YAPI

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

```
function ySetTimeout( callback, ms_timeout, arguments)
```

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

### Paramètres :

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

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

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

### Retourne :

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

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

YAPI

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

```
function ySleep( ms_duration, 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.UnregisterHub()****YAPI****yUnregisterHub()yUnregisterHub()**

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

```
function yUnregisterHub( 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.

```
function yUpdateDeviceList( 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.

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

YAPI

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

```
function yUpdateDeviceList_async( callback, context)
```

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

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

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

### Paramètres :

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

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

### Retourne :

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

## 3.2. Interface de la fonction Accelerometer

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_accelerometer.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAccelerometer = yoctolib.YAccelerometer;
php require_once('yocto_accelerometer.php');
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 renvoyée 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.

<b>accelerometer→get_highestValue()</b>	Retourne la valeur maximale observée pour l'accélération depuis le démarrage du module.
<b>accelerometer→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>accelerometer→get_logicalName()</b>	Retourne le nom logique de l'accéléromètre.
<b>accelerometer→get_lowestValue()</b>	Retourne la valeur minimale observée pour l'accélération depuis le démarrage du module.
<b>accelerometer→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>accelerometer→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>accelerometer→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>accelerometer→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>accelerometer→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>accelerometer→get_unit()</b>	Retourne l'unité dans laquelle l'accélération est exprimée.
<b>accelerometer→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>accelerometer→get_xValue()</b>	Retourne la composante X de l'accélération, sous forme de nombre à virgule.
<b>accelerometer→get_yValue()</b>	Retourne la composante Y de l'accélération, sous forme de nombre à virgule.
<b>accelerometer→get_zValue()</b>	Retourne la composante Z de l'accélération, sous forme de nombre à virgule.
<b>accelerometer→isOnline()</b>	Vérifie si le module hébergeant l'accéléromètre est joignable, sans déclencher d'erreur.
<b>accelerometer→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant l'accéléromètre est joignable, sans déclencher d'erreur.
<b>accelerometer→load(msValidity)</b>	Met en cache les valeurs courantes de l'accéléromètre, avec une durée de validité spécifiée.
<b>accelerometer→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>accelerometer→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de l'accéléromètre, avec une durée de validité spécifiée.
<b>accelerometer→nextAccelerometer()</b>	Continue l'énumération des accéléromètres commencée à l'aide de yFirstAccelerometer( ).
<b>accelerometer→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.

### 3. Reference

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

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

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

Modifie la mémoire de valeur maximale observée.

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

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

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

Modifie le nom logique de l'accéléromètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

#### **accelerometer→set(userData)**

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

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

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

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

## YAccelerometer

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

```
function yFindAccelerometer( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

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

### Retourne :

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

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

---

### **YAccelerometer**

Commence l'énumération des accéléromètres accessibles par la librairie.

```
function 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.

```
function calibrateFromPoints( rawValues, refValues)
```

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

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**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.

**function 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'accéléromètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

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

**YAccelerometer**

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

```
function 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()**  
**accelerometer→currentRawValue()**  
**accelerometer.get\_currentRawValue()**

**YAccelerometer**

---

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

```
function 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()**  
**accelerometer→currentValue()**  
**accelerometer.get\_currentValue()**

**YAccelerometer**

Retourne la valeur actuelle de l'accélération.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'accélération

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

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

**YAccelerometer**

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

**function get\_errorMessage( )**

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

**Retourne :**

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

---

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

**YAccelerometer**

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

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

**YAccelerometer**

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

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

**YAccelerometer**

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

**function 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

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

---

**YAccelerometer**

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

**function get\_functionId( )**

Par example `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()**  
**accelerometer→hardwareId()**  
**accelerometer.get\_hardwareId()**

**YAccelerometer**

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

**function get\_hardwareId( )**

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

**Retourne :**

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

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

**YAccelerometer**

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

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

**YAccelerometer**

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.

function **get\_logFrequency( )**

**Retourne :**

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

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

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

**YAccelerometer**

Retourne le nom logique de l'accéléromètre.

```
function 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()**  
**accelerometer→lowestValue()**  
**accelerometer.get\_lowestValue()**

**YAccelerometer**

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

function **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.

```
function 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\_module\_async()**  
**accelerometer→module\_async()**  
**accelerometer.get\_module\_async()**

YAccelerometer

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

**function get\_module\_async( callback, context)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

**YAccelerometer**

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

**function get\_recordedData( startTime, endTime )**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

YAccelerometer

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

function **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.

**function 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.

function **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)**  
**accelerometer→userData()**  
**accelerometer.get(userData)**

YAccelerometer

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

```
function 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.

```
function get_xValue( )
```

**Retourne :**

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

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

**accelerometer→get\_yValue()**

**YAccelerometer**

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

---

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

```
function 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.

```
function 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.

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

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

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

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

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

**function load( msValidity )**

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

**Paramètres :**

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

**Retourne :**

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

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

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

```
function loadCalibrationPoints( rawValues, refValues )
```

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

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

**YAccelerometer**

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

function **nextAccelerometer( )**

**Retourne :**

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

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

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

**function registerTimedReportCallback( **callback**)**

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

**accelerometer→registerValueCallback()**  
**accelerometer.registerValueCallback()****YAccelerometer**

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

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()`  
`accelerometer→setHighestValue()`  
`accelerometer.set_highestValue()`

---

YAccelerometer

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

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

**function set\_logFrequency( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

`accelerometer→set_logicalName()`  
`accelerometer→setLogicalName()`  
`accelerometer.set_logicalName()`

YAccelerometer

Modifie le nom logique de l'accéléromètre.

**function** `set_logicalName( newval )`

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

**Paramètres :**

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

**Retourne :**

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

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

YAccelerometer

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

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).

```
function set(userData) {
```

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

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

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

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 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()

	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>	

### 3. Reference

---

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

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

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

**anbutton→set\_logicalName(newval)**

Modifie le nom logique de l'entrée analogique.

**anbutton→set\_sensitivity(newval)**

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

**anbutton→set(userData)**

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.

```
function yFindAnButton( 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.

```
function 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()anbutton.describe()****YAnButton**

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

function **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 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).

```
function get_advertisedValue( )
```

**Retourne :**

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

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

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

**YAnButton**

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

```
function 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).

**function 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).

function **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).

**function 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.

```
function get_errorMessage( )
```

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

**Retourne :**

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

**anbutton→get\_errorType()**

**YAnButton**

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

---

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

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

**YAnButton**

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

**function 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.

**function get\_functionDescriptor( )**

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

**Retourne :**

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

---

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

---

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

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

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

**anbutton→get\_hardwareId()**

**YAnButton**

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

---

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

**function 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.

```
function 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).

function **get\_lastTimePressed( )**

**Retourne :**

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

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

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

**YAnButton**

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).

function **get\_lastTimeReleased( )**

**Retourne :**

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

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

**anbutton→get\_logicalName()**

**YAnButton**

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

---

Retourne le nom logique de l'entrée analogique.

```
function 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.

```
function 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\_module\_async()  
anbutton→module\_async()  
anbutton.get\_module\_async()****YAnButton**

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

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

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

**YAnButton**

Retourne la valeur du compteur d'impulsions.

```
function 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()** YAnButton  
**anbutton→pulseTimer()&anbutton.get\_pulseTimer()**

---

Retourne le timer du compteur d'impulsions (ms)

```
function 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).

```
function 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 déclanchement de callbacks.

```
function get_sensitivity( )
```

**Retourne :**

un entier représentant la sensibilité pour l'entrée (entre 1 et 1000) pour le déclanchement 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).

```
function 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() anbutton.isOnline()

YAnButton

---

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

```
function 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→isOnline\_async()  
anbutton.isOnline\_async()****YAnButton**

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

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

**anbutton→load()****YAnButton**

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

**function load( msValidity )**

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

**Paramètres :**

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

**Retourne :**

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

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

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

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

**YAnButton**

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

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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

```
function 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.

**function set\_analogCalibration( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

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

function **set\_calibrationMax( newval )**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

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

function **set\_calibrationMin( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

Modifie le nom logique de l'entrée analogique.

**function set\_logicalName( newval )**

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

**Paramètres :**

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

**Retourne :**

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

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

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

function **set\_sensitivity( newval )**

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

**Paramètres :**

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

**Retourne :**

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

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

**anbutton→set(userData)**

**YAnButton**

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

---

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

```
function set(userData) {
```

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

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

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

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 3.4. Interface de la fonction CarbonDioxide

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_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 renournée 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.

**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.

### 3. Reference

---

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

Modifie le nom logique du capteur de CO2.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**carbondioxide→set\_userData(data)**

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

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

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

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

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

function **yFindCarbonDioxide( 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()**

---

### **YCarbonDioxide**

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

```
function 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.

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

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

```
function calibrateFromPoints( rawValues, refValues)
```

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

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**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.

**function 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 CO2 (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

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

**YCarbonDioxide**

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

```
function get_advertisedValue( )
```

**Retourne :**

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

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

**YCarbonDioxide**

---

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

```
function 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.

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

**YCarbonDioxide**

Retourne la valeur actuelle du taux de CO2.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du taux de CO2

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

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

**YCarbonDioxide**

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

**function get\_errorMessage( )**

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

**Retourne :**

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

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

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

```
function get_errorType( )
```

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

**Retourne :**

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

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

**YCarbonDioxide**

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

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

**YCarbonDioxide**

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

**function 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

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

**YCarbonDioxide**

---

Retourne l'identifiant matériel du capteur de CO<sub>2</sub>, sans référence au module.

```
function get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le capteur de CO<sub>2</sub> (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

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

**YCarbonDioxide**

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

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

**YCarbonDioxide**

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

**function get\_highestValue( )**

**Retourne :**

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

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

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

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

**function get\_logFrequency( )****Retourne :**

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

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

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

**YCarbonDioxide**

---

Retourne le nom logique du capteur de CO2.

```
function 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()**  
**carbondioxide→lowestValue()**  
**carbondioxide.get\_lowestValue()**

**YCarbonDioxide**

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

```
function get_lowestValue( )
```

**Retourne :**

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

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

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

---

**YCarbonDioxide**

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

```
function get_module( )
```

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

**Retourne :**

une instance de `YModule`

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

**YCarbonDioxide**

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

**function get\_module\_async( callback, context)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

**function get\_recordedData( startTime, endTime )**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

```
function 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.

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

---

**YCarbonDioxide**

Retourne la résolution des valeurs mesurées.

```
function 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.

function **get\_unit( )**

**Retourne :**

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

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

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

**YCarbonDioxide**

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

```
function 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()****YCarbonDioxide**

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

**function 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

**carbon dioxide → isOnline\_async()**  
**carbon dioxide.isOnline\_async()****YCarbonDioxide**

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

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

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

**Retourne :**

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

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

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

function **load( msValidity )**

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

**Paramètres :**

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

**Retourne :**

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

**carbon dioxide → loadCalibrationPoints()**  
**carbon dioxide.loadCalibrationPoints()****YCarbonDioxide**

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

```
function loadCalibrationPoints( rawValues, refValues )
```

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

**carbon dioxide → nextCarbonDioxide()**  
**carbon dioxide.nextCarbonDioxide()**

---

**YCarbonDioxide**

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

```
function nextCarbonDioxide( )
```

**Retourne :**

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

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

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

```
function registerTimedReportCallback( callback)
```

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

**Paramètres :**

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

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

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

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**carbondioxide→setHighestValue()**  
**carbondioxide.set\_highestValue()**

**YCarbonDioxide**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

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

**function set\_logFrequency( newval )**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

Modifie le nom logique du capteur de CO2.

```
function set_logicalName( newval)
```

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

**Paramètres :**

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

**Retourne :**

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

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

---

**YCarbonDioxide**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval )
```

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

---

**YCarbonDioxide**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

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

```
function set(userData) {
```

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

**carbondioxide→wait\_async()**  
**carbondioxide.wait\_async()****YCarbonDioxide**

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

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 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>
node.js	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.

### 3. Reference

#### **colorled→get\_module()**

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

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

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

#### **colorled→get\_rgbColor()**

Retourne la couleur RGB courante de la led.

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

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

#### **colorled→get\_userData()**

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

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

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

#### **colorled→isOnline()**

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

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

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

#### **colorled→load(msValidity)**

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

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

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

#### **colorled→nextColorLed()**

Continue l'énumération des leds RGB commencée à l'aide de yFirstColorLed( ).

#### **colorled→registerValueCallback(callback)**

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

#### **colorled→rgbMove(rgb\_target, ms\_duration)**

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

#### **colorled→set\_hslColor(newval)**

Modifie la couleur courante de la led, en utilisant une couleur HSL spécifiée.

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

Modifie le nom logique de la led RGB.

#### **colorled→set\_rgbColor(newval)**

Modifie la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu).

#### **colorled→set\_rgbColorAtPowerOn(newval)**

Modifie la couleur que la led va afficher spontanément à l'allumage du module.

#### **colorled→set\_userData(data)**

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

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

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

## YColorLed.FindColorLed() yFindColorLed()yFindColorLed()

**YColorLed**

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

```
function yFindColorLed( 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.

```
function 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()colorled.describe()****YColorLed**

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

```
function 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 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).

```
function 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.

```
function 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.

```
function get_errorType( )
```

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

**Retourne :**

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

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

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

```
function 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.

**function get\_functionDescriptor( )**

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

**Retourne :**

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

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

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

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

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

**colorled→get.hardwareId()**

**YColorLed**

**colorled→hardwareId()colorled.get.hardwareId()**

---

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

```
function 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.

```
function get_hslColor( )
```

**Retourne :**

un entier représentant la couleur HSL courante de la led

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

**colorled→get\_logicalName()**

**YColorLed**

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

---

Retourne le nom logique de la led RGB.

```
function 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.

```
function 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\_module\_async()**  
**colorled→module\_async()**  
**colorled.get\_module\_async()****YColorLed**

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

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

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

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

**Paramètres :**

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

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

**Retourne :**

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

**colorled→get\_rgbColor()****YColorLed****colorled→rgbColor()colorled.get\_rgbColor()**

Retourne la couleur RGB courante de la led.

```
function 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()**  
**colorled→rgbColorAtPowerOn()**  
**colorled.get\_rgbColorAtPowerOn()**

---

**YColorLed**

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

```
function 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).

```
function get(userData) {
```

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

**colorled→hsIMove()colorled.hsiMove()****YColorLed**

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

```
function hsiMove( hsl_target, ms_duration)
```

**Paramètres :**

**hsl\_target** couleur HSL désirée à la fin de la transition

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

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

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

**colorled→isOnline()colorled.isOnline()****YColorLed**

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

```
function 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→isOnline\_async()colorled.isOnline\_async()****YColorLed**

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

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

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

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

```
function load( msValidity)
```

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

**Paramètres :**

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

**Retourne :**

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

**colorled→load\_async()colorled.load\_async()****YColorLed**

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

**colorled→nextColorLed()colorled.nextColorLed()****YColorLed**

Continue l'énumération des leds RGB commencée à l'aide de `yFirstColorLed()`.

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()colorled.rgbMove()****YColorLed**

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

```
function rgbMove( rgb_target, ms_duration)
```

**Paramètres :**

**rgb\_target** couleur RGB désirée à la fin de la transition

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

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

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

**colorled→set\_hslColor()**

**YColorLed**

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

---

Modifie la couleur courante de la led, en utilisant une couleur HSL spécifiée.

```
function set_hslColor( 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.

```
function set_logicalName( newval)
```

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

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique de la led RGB.

**Retourne :**

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

**colorled→set\_rgbColor()**

**YColorLed**

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

---

Modifie la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu).

```
function set_rgbColor( newval)
```

L'encodage est réalisé de la manière suivante: 0xRRGGBB.

**Paramètres :**

**newval** un entier représentant la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu)

**Retourne :**

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

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

**colorled→set\_rgbColorAtPowerOn()**  
**colorled→setRgbColorAtPowerOn()**  
**colorled.set\_rgbColorAtPowerOn()**

**YColorLed**

Modifie la couleur que la led va afficher spontanément à l'allumage du module.

**function set\_rgbColorAtPowerOn( newval)**

Cette couleur sera affichée dès que le module sera sous tension. Ne pas oublier d'appeler la fonction `saveToFlash()` du module correspondant pour que ce paramètre soit mémorisé.

**Paramètres :**

**newval** un entier représentant la couleur que la led va afficher spontanément à l'allumage du module

**Retourne :**

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

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

**colorled→set(userData)**

**YColorLed**

**colorled→setUserData()|colorled.set(userData)**

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

```
function set(userData) {
```

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

**colorled→wait\_async()colorled.wait\_async()****YColorLed**

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

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 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 rentrée par le capteur (sans arrondi ni calibration).

#### compass→get\_currentValue()

Retourne la valeur actuelle du cap relatif.

#### compass→get\_errorMessage()

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

#### compass→get\_errorType()

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

#### compass→get\_friendlyName()

Retourne un identifiant global du compas au format NOM\_MODULE.NOM\_FONCTION.

#### compass→get\_functionDescriptor()

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

#### compass→get\_functionId()

Retourne l'identifiant matériel du compas, sans référence au module.

#### compass→get\_hardwareId()

Retourne l'identifiant matériel unique du compas au format SERIAL.FUNCTIONID.

**compass→get\_highestValue()**

Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.

**compass→get\_logFrequency()**

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

**compass→get\_logicalName()**

Retourne le nom logique du compas.

**compass→get\_lowestValue()**

Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.

**compass→get\_magneticHeading()**

Retourne la direction du nord magnétique, indépendamment du cap configuré.

**compass→get\_module()**

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

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

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

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

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

**compass→get\_reportFrequency()**

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

**compass→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**compass→get\_unit()**

Retourne l'unité dans laquelle le cap relatif est exprimée.

**compass→get\_userData()**

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

**compass→isOnline()**

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

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

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

**compass→load(msValidity)**

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

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

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

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

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

**compass→nextCompass()**

Continue l'énumération des compas commencée à l'aide de yFirstCompass( ).

**compass→registerTimedReportCallback(callback)**

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

**compass→registerValueCallback(callback)**

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

**compass→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

### 3. Reference

---

**compass→set\_logFrequency(newval)**

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

**compass→set\_logicalName(newval)**

Modifie le nom logique du compas.

**compass→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**compass→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**compass→set\_userData(data)**

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

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

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

**YCompass.FindCompass()****yFindCompass()yFindCompass()****YCompass**

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

**function yFindCompass( 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() yFirstCompass()yFirstCompass()**

---

**YCompass**

Commence l'énumération des compas accessibles par la librairie.

```
function 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.

```
function calibrateFromPoints( rawValues, refValues)
```

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

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

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

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

**Retourne :**

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

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

**compass→describe()compass.describe()****YCompass**

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

```
function 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 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).

```
function 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 renvoyée par le capteur (sans arrondi ni calibration).

function **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**.

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

**YCompass**

Retourne la valeur actuelle du cap relatif.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du cap relatif

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

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

**YCompass**

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

**function get\_errorMessage( )**

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

**Retourne :**

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

**compass→get\_errorType()****YCompass****compass→errorType()compass.get\_errorType()**

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

```
function 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.

**function 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.

**function 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

**compass→get\_functionId()**

**YCompass**

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

---

Retourne l'identifiant matériel du compas, sans référence au module.

**function 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.

```
function 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.

**function 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.

function **get\_logFrequency( )**

**Retourne :**

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

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

**compass→get\_logicalName()**

**YCompass**

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

---

Retourne le nom logique du compas.

```
function get_logicalName( )
```

**Retourne :**

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

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

Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.

```
function 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.

**function 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.

```
function 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\_module\_async()**  
**compass→module\_async()**  
**compass.get\_module\_async()**

**YCompass**

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

**function get\_module\_async( callback, context)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

**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.

**function get\_recordedData( startTime, endTime)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

**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.

function **get\_reportFrequency( )**

**Retourne :**

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

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

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

Retourne la résolution des valeurs mesurées.

```
function 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.

```
function 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).

```
function 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.

```
function 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→isOnline\_async()**  
**compass.isOnline\_async()****YCompass**

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

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

**compass→load()compass.load()****YCompass**

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

**function load( msValidity )**

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

**Paramètres :**

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

**Retourne :**

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

**compass→loadCalibrationPoints()**  
**compass.loadCalibrationPoints()****YCompass**

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

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

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

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

**Retourne :**

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

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

**compass→load\_async()compass.load\_async()****YCompass**

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

**compass→nextCompass()compass.nextCompass()****YCompass**

Continue l'énumération des compas commencée à l'aide de `yFirstCompass()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**compass→setHighestValue()**  
**compass.set\_highestValue()**

---

**YCompass**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

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

**function set\_logFrequency( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie le nom logique du compas.

**function set\_logicalName( newval )**

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

**Paramètres :**

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

**Retourne :**

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

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

**YCompass**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval )
```

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

**Paramètres :**

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

**Retourne :**

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

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

**compass→set(userData)**

**YCompass**

**compass→setUserData()compass.set(userData)**

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

```
function set(userData) {
```

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

**compass→wait\_async()compass.wait\_async()****YCompass**

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

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 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>
nodejs var yoctolib = require('yoctolib');
var YCurrent = yoctolib.YCurrent;
require_once('yocto_current.php');
#include "yocto_current.h"
m #import "yocto_current.h"
pas uses yocto_current;
vb yocto_current.vb
cs yocto_current.cs
java import com.yoctopuce.YoctoAPI.YCurrent;
py from yocto_current import *

```

### Fonction globales

#### **yFindCurrent(func)**

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

#### **yFirstCurrent()**

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

### Méthodes des objets YCurrent

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

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

#### **current→describe()**

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

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

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

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

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

#### **current→get\_currentValue()**

Retourne la valeur instantanée du courant.

#### **current→get\_errorMessage()**

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

#### **current→get\_errorType()**

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

#### **current→get\_friendlyName()**

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

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

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

#### **current→get\_functionId()**

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

#### **current→get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de courant au format SERIAL.FUNCTIONID.
<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>

### 3. Reference

---

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

**current→set\_logicalName(newval)**

Modifie le nom logique du capteur de courant.

**current→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour le courant.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**current→set\_resolution(newval)**

Modifie la résolution des valeurs mesurées.

**current→set\_userData(data)**

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

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

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

**YCurrent.FindCurrent()****YCurrent****yFindCurrent()yFindCurrent()**

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

**function yFindCurrent( 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() yFirstCurrent()yFirstCurrent()**

**YCurrent**

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

```
function 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.

```
function calibrateFromPoints( rawValues, refValues)
```

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

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

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

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

**Retourne :**

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

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

**current→describe()current.describe()****YCurrent**

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

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

**YCurrent**

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

```
function 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 renvoyée par le capteur (sans arrondi ni calibration).

```
function 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.

---

<b>current→get_currentValue()</b>	<b>YCurrent</b>
<b>current→currentValue()current.get_currentValue()</b>	

---

Retourne la valeur instantanée du courant.

```
function 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()**

**YCurrent**

**current→errorMessage()current.getErrorMessage()**

---

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

**function 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.

```
function get_errorType( )
```

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

**Retourne :**

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

**current→get\_friendlyName()**

**YCurrent**

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

---

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

```
function 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.

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

**YCurrent**

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

**function 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

**current→get\_functionId()**

**YCurrent**

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

---

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

function **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.

```
function 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()**

**YCurrent**

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

---

Retourne la valeur maximale observée pour le courant.

```
function get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le courant

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

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

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

```
function get_logFrequency( )
```

**Retourne :**

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

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

**current→get\_logicalName()**

**YCurrent**

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

---

Retourne le nom logique du capteur de courant.

```
function get_logicalName( )
```

**Retourne :**

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

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

Retourne la valeur minimale observée pour le courant.

```
function 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.

```
function 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\_module\_async()**  
**current→module\_async()**  
**current.get\_module\_async()**

YCurrent

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

**function get\_module\_async( callback, context)**

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

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

**Paramètres :**

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

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

**Retourne :**

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

---

<b>current→get_recordedData()</b>	<b>YCurrent</b>
<b>current→recordedData()current.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.

**function get\_recordedData( startTime, endTime )**

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

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

**Paramètres :**

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

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

**Retourne :**

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

**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.

```
function 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.

```
function 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.

```
function 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).

```
function 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()current.isOnline()****YCurrent**

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

```
function 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→isOnline\_async()current.isOnline\_async()****YCurrent**

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

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

**current→load()current.load()****YCurrent**

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

```
function load( msValidity)
```

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

**Paramètres :**

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

**Retourne :**

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

**current→loadCalibrationPoints()  
current.loadCalibrationPoints()****YCurrent**

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

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

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

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

**Retourne :**

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

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

**current→load\_async()current.load\_async()****YCurrent**

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

## current→nextCurrent()current.nextCurrent()

YCurrent

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

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

**Paramètres :**

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

**current→registerValueCallback()  
current.registerValueCallback()****YCurrent**

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

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour le courant

**Retourne :**

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

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

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

**YCurrent**

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

**function set\_logFrequency( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

---

<b>current→set_logicalName()</b>	<b>YCurrent</b>
<b>current→setLogicalName()current.set_logicalName()</b>	

---

Modifie le nom logique du capteur de courant.

```
function set_logicalName( newval)
```

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

**Paramètres :**

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

**Retourne :**

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

**current→set\_lowestValue()**

**YCurrent**

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

---

Modifie la mémoire de valeur minimale observée pour le courant.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour le courant

**Retourne :**

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

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

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

**YCurrent**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

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

**Paramètres :**

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

**Retourne :**

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

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

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

---

Modifie la résolution des valeurs mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

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

**Retourne :**

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

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

**current→set(userData)****YCurrent****current→setUserData()current.set(userData)**

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

```
function set(userData) {
```

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

**current→wait\_async()current.wait\_async()****YCurrent**

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

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 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>
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 *

```

### 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()

### 3. Reference

Retourne un identifiant global de l'enregistreur de données au format NOM_MODULE . NOM_FONCTION.
<b>datalogger→get_functionDescriptor()</b> Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>datalogger→get_functionId()</b> Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.
<b>datalogger→get_hardwareId()</b> Retourne l'identifiant matériel unique de l'enregistreur de données au format SERIAL . FUNCTIONID.
<b>datalogger→get_logicalName()</b> Retourne le nom logique de l'enregistreur de données.
<b>datalogger→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>datalogger→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>datalogger→get_recording()</b> Retourne l'état d'activation de l'enregistreur de données.
<b>datalogger→get_timeUTC()</b> Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.
<b>datalogger→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>datalogger→isOnline()</b> Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.
<b>datalogger→isOnline_async(callback, context)</b> Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.
<b>datalogger→load(msValidity)</b> Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.
<b>datalogger→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.
<b>datalogger→nextDataLogger()</b> Continue l'énumération des enregistreurs de données commencée à l'aide de yFirstDataLogger( ).
<b>datalogger→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>datalogger→set_autoStart(newval)</b> Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.
<b>datalogger→set_logicalName(newval)</b> Modifie le nom logique de l'enregistreur de données.
<b>datalogger→set_recording(newval)</b> Modifie l'état d'activation de l'enregistreur de données.
<b>datalogger→set_timeUTC(newval)</b> Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.
<b>datalogger→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>datalogger→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.

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

## YDataLogger

Permet de retrouver un enregistreur de données d'après un identifiant donné.

**function yFindDataLogger( 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.

```
function 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()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.

function **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'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.

```
function 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).

```
function get_advertisedValue( )
```

**Retourne :**

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

**datalogger→get\_autoStart()**

**YDataLogger**

**datalogger→autoStart()datalogger.get\_autoStart()**

---

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

**function 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.

function **get\_currentRunIndex( )**

**Retourne :**

un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

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

**datalogger→get\_dataSets()**

**YDataLogger**

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

---

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

**function 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).

**function get\_dataStreams( v )**

L'appelant doit passer par référence un tableau vide pour stocker les objets YDataStream, et la méthode va les remplir avec des objets décrivant les séquences de données disponibles.

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

**Paramètres :**

**v** un tableau de YDataStreams qui sera rempli avec les séquences trouvées

**Retourne :**

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

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

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

**YDataLogger**

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

**function get\_errorMessage( )**

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

**Retourne :**

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

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

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

```
function 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()**  
**datalogger→friendlyName()**  
**datalogger.get\_friendlyName()**

**YDataLogger**

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

```
function get_friendlyName( )
```

Le chaîne renvoyée 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.

**function 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

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

**YDataLogger**

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

function **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.

```
function 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.

**function 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.

```
function 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\_module\_async()**  
**datalogger→module\_async()**  
**datalogger.get\_module\_async()**

**YDataLogger**

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

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

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

```
function get_recording( )
```

**Retourne :**

soit Y\_RECORDING\_OFF, soit Y\_RECORDING\_ON, selon l'état d'activation de l'enregistreur de données

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

**datalogger→get\_timeUTC()**

**YDataLogger**

**datalogger→timeUTC()datalogger.get\_timeUTC()**

---

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

```
function get_timeUTC( )
```

**Retourne :**

un entier représentant le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue

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

**datalogger→get(userData)****YDataLogger****datalogger→userData()datalogger.get(userData)**

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

```
function 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()datalogger.isOnline()****YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()  
datalogger.isOnline\_async()****YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

**datalogger→load()datalogger.load()****YDataLogger**

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

**function load( msValidity)**

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

**Paramètres :**

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

**Retourne :**

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

**datalogger→load\_async()datalogger.load\_async()****YDataLogger**

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

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

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

**Retourne :**

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

**datalogger→nextDataLogger()**  
**datalogger.nextDataLogger()**

---

**YDataLogger**

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

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**

**YDataLogger**

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

---

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

function **set\_autoStart( newval )**

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

**Paramètres :**

**newval** soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

**Retourne :**

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

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

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

**YDataLogger**

Modifie le nom logique de l'enregistreur de données.

```
function set_logicalName( newval )
```

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

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique de l'enregistreur de données.

**Retourne :**

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

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

**YDataLogger**

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

function **set\_recording( newval )**

**Paramètres :**

**newval** soit Y\_RECORDING\_OFF, soit Y\_RECORDING\_ON, selon l'état d'activation de l'enregistreur de données

**Retourne :**

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

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

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

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

```
function set_timeUTC( newval)
```

**Paramètres :**

**newval** un entier représentant la référence de temps UTC, afin de l'attacher aux données enregistrées

**Retourne :**

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

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

**datalogger→set(userData)**

**YDataLogger**

**datalogger→setUserData()datalogger.set(userData)**

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

```
function set(userData) {
```

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

**datalogger→wait\_async()datalogger.wait\_async()****YDataLogger**

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

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

**Retourne :**

rien du tout :

## 3.9. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_datalogger.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDataLogger = yoctolib.YDataLogger;
php require_once('yocto_datalogger.php');
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\_averageValue()** YDataRun  
**datarun→averageValue()datarun.get\_averageValue()**

Retourne la valeur moyenne des mesures observées au moment choisi.

```
function get_averageValue( measureName, pos)
```

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

Retourne la valeur moyenne des mesures observées au moment choisi.

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

**Paramètres :**

**measureName** le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

**pos** l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

**Retourne :**

une nombre flottant (la valeur moyenne).

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

**datarun→get\_duration()****YDataRun****datarun→duration()datarun.get\_duration()**

Retourne la durée (en secondes) du Run.

```
function get_duration( )
```

**datarun→get\_duration()****datarun→duration()datarun.get\_duration()**

Retourne la durée (en secondes) du Run.

```
js   function get_duration( )  
nodejs function get_duration( )  
php  function get_duration( )  
java long get_duration( )  
py   def get_duration( )
```

Lorsque cette méthode est appellée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le début du Run (quand le module a été mis sous tension) et la dernière mesure enregistrée.

**datarun→get\_maxValue()****YDataRun****datarun→maxValue()datarun.get\_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

```
function get_maxValue( measureName, pos)
```

**datarun→get\_maxValue()****datarun→maxValue()datarun.get\_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

```
[js] function get_maxValue( measureName, pos)
```

```
[nodejs] function get_maxValue( measureName, pos)
```

```
[php] function get_maxValue( $measureName, $pos)
```

```
[java] double get_maxValue( String measureName, int pos)
```

```
[py] def get_maxValue( measureName, pos)
```

**Paramètres :**

**measureName** le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

**pos** l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

**Retourne :**

une nombre flottant (la valeur maximale).

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

**datarun→get\_measureNames()**  
**datarun→measureNames()**  
**datarun.get\_measureNames()**

**YDataRun**

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

**function get\_measureNames( )**

**datarun→get\_measureNames()**  
**datarun→measureNames()datarun.get\_measureNames()**

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

**js** function **get\_measureNames( )**  
**nodejs** function **get\_measureNames( )**  
**php** function **get\_measureNames( )**  
**java** ArrayList<String> **get\_measureNames( )**  
**py** def **get\_measureNames( )**

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure.

**Retourne :**

une liste de chaîne de caractères (les noms des mesures)

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

**datarun→get\_minValue()****YDataRun****datarun→minValue()datarun.get\_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

```
function get_minValue( measureName, pos)
```

**datarun→get\_minValue()****datarun→minValue()datarun.get\_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

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

**Paramètres :**

**measureName** le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

**pos** l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

**Retourne :**

une nombre flottant (la valeur minimale).

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

**datarun→get\_startTimeUTC()**  
**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.

**datarun→get\_valueCount()****YDataRun****datarun→valueCount()datarun.get\_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

```
function get_valueCount( )
```

**datarun→get\_valueCount()****datarun→valueCount()datarun.get\_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

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

Lorsque cette méthode est appellée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargeement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

**Retourne :**

un entier positif correspondant à la durée du Run divisée par l'intervalle entre les valeurs.

**datarun→get\_valueInterval()****YDataRun****datarun→valueInterval()datarun.get\_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

```
function get_valueInterval( )
```

**datarun→get\_valueInterval()****datarun→valueInterval()datarun.get\_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

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

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

**Retourne :**

un entier positif correspondant au nombre de secondes couvertes par chaque valeur représentée dans le Run.

**datarun→set\_valueInterval()**  
**datarun→setValueInterval()**  
**datarun.set\_valueInterval()**

YDataRun

Change l'intervalle de temps représenté par chaque valeur de ce run.

function **set\_valueInterval( valueInterval)**

**datarun→set\_valueInterval()**  
**datarun→setValueInterval()datarun.set\_valueInterval()**

Change l'intervalle de temps représenté par chaque valeur de ce run.

**js** function **set\_valueInterval( valueInterval)**  
**nodejs** function **set\_valueInterval( valueInterval)**  
**php** function **set\_valueInterval( \$valueInterval)**  
**java** void **set\_valueInterval( int valueInterval)**  
**py** def **set\_valueInterval( valueInterval)**

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

**Paramètres :**

**valueInterval** un nombre entier de secondes.

**Retourne :**

nothing

## 3.10. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-même sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <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()`

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

**dataset→loadMore()**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

**dataset→loadMore\_async(callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**dataset→get\_endTimeUTC()****YDataSet****dataset→endTimeUTC()dataset.get\_endTimeUTC()**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

**function 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.

```
function 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.

**function get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple THRMCP11-123456.temperature1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: THRMCP11-123456.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.

```
function 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.

**function 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.

```
function 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→startTimeUTC()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).

**function 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.

```
function 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.

**function 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.

```
function 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.

**dataset→loadMore\_async()dataset.loadMore\_async()****YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
function loadMore_async( callback, context)
```

**Paramètres :**

**callback** fonction fournie par l'utilisateur, qui sera appelée lorsque la suite du chargement aura été effectué. La fonction callback doit prendre trois arguments: - la variable de contexte à disposition de l'utilisateur - l'objet YDataSet dont la méthode loadMore\_async a été appelée - le résultat de l'appel: soit l'état d'avancement du chargement (0...100), ou un code d'erreur négatif en cas de problème.

**context** variable de contexte à disposition de l'utilisateur

**Retourne :**

rien.

## 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.

**function 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.

**function 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.

**function 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()  
datastream→data()datastream.get\_data()****YDataStream**

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

**function get\_data( row, col)**

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode 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()****YDataStream****datastream→dataRows()datastream.get\_dataRows()**

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

```
function 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.

**function get\_dataSamplesIntervalMs( )**

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la fréquence d'enregistrement peut être changée pour chaque fonction.

**Retourne :**

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

---

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

---

Retourne la durée approximative de cette séquence, en secondes.**function get\_duration( )****Retourne :**

le nombre de secondes couvertes par cette séquence.

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

**datastream→get\_maxValue()**

**YDataStream**

**datastream→maxValue()datastream.get\_maxValue()**

---

Retourne la plus grande valeur observée durant cette séquence.

**function get\_maxValue( )**

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus grande valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

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

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

Retourne la plus petite valeur observée durant cette séquence.

```
function 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()**

**YDataStream**

**datastream→rowCount()datastream.getRowCount()**

---

Retourne le nombre d'enregistrement contenus dans la séquence.

**function 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.

```
function 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()****YDataStream****datastream→startTime()datastream.getStartTime()**

Retourne le temps de départ relatif de la séquence (en secondes).

**function 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).

function **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.

**function yFindDigitalIO( 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.

```
function 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.

```
function delayedPulse( bitno, ms_delay, ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**ms\_delay** délai d'attente avant l'impulsion, en millisecondes

**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

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

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

**digitalio→describe()digitalio.describe()****YDigitalIO**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**function 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 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).

```
function get_advertisedValue( )
```

**Retourne :**

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

**digitalio→get\_bitDirection()** YDigitalIO  
**digitalio→bitDirection()digitalio.get\_bitDirection()**

---

Retourne la direction d'un seul bit du port d'E/S.

```
function get_bitDirection( bitno )
```

(0 signifie que le bit est une entrée, 1 une sortie)

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

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

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

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

Retourne la direction d'un seul bit du port d'E/S.

```
function get_bitOpenDrain( bitno )
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert)..

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

**digitalio→get\_bitPolarity()**

**YDigitalIO**

**digitalio→bitPolarity()digitalio.get\_bitPolarity()**

---

Retourne la polarité d'un seul bit du port d'E/S.

```
function get_bitPolarity( bitno )
```

0 signifie que l'entrée sortie est en mode normal, 1 qu'elle est en mode inverse

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

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

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

**digitalio→get\_bitState()****YDigitalIO****digitalio→bitState()digitalio.get\_bitState()**

Retourne l'état d'un seul bit du port d'E/S.

```
function get_bitState( bitno )
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

l'état du bit (0 ou 1).

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

**digitalio→get\_errorMessage()**  
**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.

**function get\_errorMessage( )**

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

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

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

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
function get_errorType( )
```

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

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

## **digitalio→get\_friendlyName()**

**YDigitalIO**

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

---

Retourne un identifiant global du port d'E/S digital au format NOM\_MODULE . NOM\_FONCTION.

```
function 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.

```
function 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()**

**YDigitalIO**

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

---

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

**function get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**digitalio→get\_hardwareId()****YDigitalIO****digitalio→hardwareId()digitalio.get\_hardwareId()**

Retourne l'identifiant matériel unique du port d'E/S digital au format SERIAL.FUNCTIONID.

```
function 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()**

**YDigitalIO**

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

---

Retourne le nom logique du port d'E/S digital.

```
function 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.

```
function 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\_module\_async()**  
**digitalio→module\_async()**  
**digitalio.get\_module\_async()**

YDigitalIO

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

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

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

Retourne la source de tension utilisée pour piloter les bits en sortie.

```
function get_outputVoltage( )
```

**Retourne :**

une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

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

**digitalio→get\_portDirection()**

**YDigitalIO**

**digitalio→portDirection()digitalio.get\_portDirection()**

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
function 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).

```
function get_portOpenDrain( )
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

**Retourne :**

un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

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

**digitalio→get\_portPolarity()**

**YDigitalIO**

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

---

Retourne la polarité des bits du port (bitmap).

```
function 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.

```
function 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.

**function 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).

```
function 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()digitalio.isOnline()****YDigitalIO**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()digitalio.isOnline\_async()****YDigitalIO**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

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

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

**Retourne :**

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

**digitalio→load()  
digitalio.load()****YDigitalIO**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

**function load( msValidity )**

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

**Paramètres :**

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

**Retourne :**

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

## digitalio→load\_async()**digitalio.load\_async()**

YDigitalIO

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

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

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

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

### Retourne :

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

## **digitalio→nextDigitalIO()digitalio.nextDigitalIO()**

**YDigitalIO**

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

```
function 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.

```
function pulse( bitno, ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

### Paramètres :

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

### Retourne :

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

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

**digitalio→registerValueCallback()  
digitalio.registerValueCallback()****YDigitalIO**

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

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()****YDigitalIO****digitalio→setBitDirection()digitalio.set\_bitDirection()**

Change la direction d'un seul bit du port d'E/S.

```
function set_bitDirection( bitno, bitdirection)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitdirection** nouvelle valeur de la direction, 0=entrée, 1=sortie. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

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

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

**digitalio→set\_bitOpenDrain()**  
**digitalio→setBitOpenDrain()**  
**digitalio.set\_bitOpenDrain()**

YDigitalIO

Change le type d'interface électrique d'un seul bit du port d'E/S.

```
function set_bitOpenDrain( bitno, opendrain)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**opendrain** 0 pour faire une entrée ou une sortie digitale standard, 1 pour une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

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

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

**digitalio→set\_bitPolarity()****YDigitalIO****digitalio→setBitPolarity()digitalio.set\_bitPolarity()**

Change la polarité d'un seul bit du port d'E/S.

```
function set_bitPolarity( bitno, bitpolarity)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitpolarity** nouvelle valeur de la polarité. 0=mode normal, 1=mode inverse. N'oubliez pas d'appeler la méthode saveToFlash() du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

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

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

**digitalio→set\_bitState()**

**YDigitalIO**

**digitalio→setBitState()digitalio.set\_bitState()**

---

Change l'état d'un seul bit du port d'E/S.

```
function set_bitState( bitno, bitstate)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitstate** nouvel état du bit (1 ou 0)

**Retourne :**

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

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

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

**YDigitalIO**

Modifie le nom logique du port d'E/S digital.

```
function set_logicalName( newval)
```

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

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du port d'E/S digital.

**Retourne :**

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

**digitalio→set\_outputVoltage()**  
**digitalio→setOutputVoltage()**  
**digitalio.set\_outputVoltage()**

**YDigitalIO**

Modifie la source de tension utilisée pour piloter les bits en sortie.

```
function set_outputVoltage( newval)
```

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

**Paramètres :**

**newval** une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

**Retourne :**

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

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

**digitalio→set\_portDirection()**  
**digitalio→setPortDirection()**  
**digitalio.set\_portDirection()**

YDigitalIO

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

function **set\_portDirection( newval)**

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

**Paramètres :**

**newval** un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

**Retourne :**

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

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

**digitalio→set\_portOpenDrain()**  
**digitalio→setPortOpenDrain()**  
**digitalio.set\_portOpenDrain()**

**YDigitalIO**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

**function set\_portOpenDrain( newval)**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

**Retourne :**

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

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

**digitalio→set\_portPolarity()****YDigitalIO****digitalio→setPortPolarity()digitalio.set\_portPolarity()**

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

```
function set_portPolarity( newval)
```

**Paramètres :**

**newval** un entier représentant la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portState()**

**YDigitalIO**

**digitalio→setPortState()digitalio.set\_portState()**

---

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
function set_portState( newval)
```

Seuls les bits configurés en sortie dans portDirection sont affectés.

**Paramètres :**

**newval** un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set(userData)****YDigitalIO****digitalio→setUserData()digitalio.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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.

```
function toggle_bitState( bitno)
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→wait\_async()digitalio.wait\_async()**

YDigitalIO

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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() yFindDisplay()yFindDisplay()

YDisplay

Permet de retrouver un ecran d'après un identifiant donné.

```
function yFindDisplay( 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() yFirstDisplay()yFirstDisplay()

YDisplay

Commence l'énumération des écran accessibles par la librairie.

```
function 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.

```
function copyLayerContent( srcLayerId, dstLayerId)
```

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**srcLayerId** l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

**dstLayerId** l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→describe()display.describe()****YDisplay**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function 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.

```
function fade( brightness, duration)
```

### Paramètres :

**brightness** nouvelle valeur de luminosité de l'écran

**duration** durée en millisecondes de la transition.

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→get\_advertisedValue()**  
**display→advertisedValue()**  
**display.get\_advertisedValue()**

**YDisplay**

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

```
function 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).

```
function get_brightness( )
```

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne Y\_BRIGHTNESS\_INVALID.

**display→get\_displayHeight()**

**YDisplay**

**display→displayHeight()display.get\_displayHeight()**

---

Retourne la hauteur de l'écran, en pixels.

```
function get_displayHeight( )
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYHEIGHT\_INVALID.

**display→get\_displayLayer()****YDisplay****display→displayLayer()display.get\_displayLayer()**

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

```
function get_displayLayer( layerId )
```

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

**Paramètres :**

**layerId** l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

**Retourne :**

un objet YDisplayLayer

En cas d'erreur, déclenche une exception ou retourne null.

**display→get\_displayType()**

**YDisplay**

**display→displayType()display.get\_displayType()**

---

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

```
function get_displayType( )
```

**Retourne :**

une valeur parmi Y\_DISPLAYTYPE\_MONO, Y\_DISPLAYTYPE\_GRAY et Y\_DISPLAYTYPE\_RGB  
représentant le type de l'écran: monochrome, niveaux de gris ou couleur

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYTYPE\_INVALID.

**display→get\_displayWidth()****YDisplay****display→displayWidth()display.get\_displayWidth()**

Retourne la largeur de l'écran, en pixels.

```
function 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.

```
function 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()****YDisplay****display→errorMessage()display.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
function 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.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

**display→get\_friendlyName()****YDisplay****display→friendlyName()display.get\_friendlyName()**

Retourne un identifiant global de l'écran au format NOM\_MODULE.NOM\_FONCTION.

```
function 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()</b>	
<b>display.get_functionDescriptor()</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function 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.

```
function 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.

**function get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'écran (par exemple RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'écran (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**display→get\_layerCount()  
display→layerCount()display.get\_layerCount()****YDisplay**

Retourne le nombre des couches affichables disponibles.

```
function 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.

```
function 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.

```
function 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.

```
function 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()  
display→module()display.get\_module()****YDisplay**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function 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\_module\_async()**  
**display→module\_async()**  
**display.get\_module\_async()****YDisplay**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**display→get\_orientation()****YDisplay****display→orientation()display.get\_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

```
function 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.

```
function 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).

```
function 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()**display.isOnline()**

YDisplay

---

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()display.isOnline\_async()****YDisplay**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## display→load()**display.load()**

## YDisplay

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
function load( msValidity )
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## display→load\_async()display.load\_async()

## YDisplay

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**display→newSequence()display.newSequence()****YDisplay**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

```
function 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()`.

```
function 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.

```
function pauseSequence( delay_ms )
```

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

**Paramètres :**

**delay\_ms** la durée de l'attente, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→playSequence()  
display.playSequence()****YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes newSequence( ) et saveSequence( ).

```
function playSequence( sequenceName)
```

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→registerValueCallback()  
display.registerValueCallback()****YDisplay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function 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.

```
function saveSequence( sequenceName )
```

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**display→set\_brightness()**  
**display→setBrightness()display.set\_brightness()****YDisplay**

Modifie la luminositéde l'écran.

```
function set_brightness( newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la 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.

```
function set_enabled( newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état d'activité de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**display→set\_logicalName()** **YDisplay**  
**display→setLogicalName()display.set\_logicalName()**

---

Modifie le nom logique de l'écran.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'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()** **YDisplay**  
**display→setOrientation()display.set\_orientation()**

Modifie l'orientation de l'écran.

```
function set_orientation( newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP,  
Y\_ORIENTATION\_RIGHT et Y\_ORIENTATION\_DOWN représentant l'orientation de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_startupSeq()****YDisplay****display→setStartupSeq()display.set\_startupSeq()**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
function set_startupSeq( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set(userData)** YDisplay  
**display→setUserData()display.set(userData)**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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.

```
function stopSequence( )
```

L'affichage est laissé tel quel.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→swapLayerContent()  
display.swapLayerContent()****YDisplay**

Permute le contenu de deux couches d'affichage.

```
function swapLayerContent( layerIdA, layerIdB)
```

La couleur et la transparence de tous les pixels des deux couches sont permutees. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**layerIdA** l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

**layerIdB** l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→upload()**display.upload()******YDisplay**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

```
function upload( pathname, content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→wait\_async()display.wait\_async()****YDisplay**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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()displaylayer.clear()****YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

**function 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.

```
function 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.

function **consoleOut( text)**

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

**Paramètres :**

**text** le message à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBar()displaylayer.drawBar()****YDisplayLayer**

Dessine un rectangle plein à une position spécifiée.

```
function drawBar( x1, y1, x2, y2)
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

**y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

**x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

**y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBitmap()  
displaylayer.drawBitmap()****YDisplayLayer**

Dessine un bitmap à la position spécifiée de la couche.

```
function drawBitmap( x, y, w, bitmap, bgcol)
```

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour 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.

```
function drawCircle( x, y, r)
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle

**y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle

**r** le rayon du cercle, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawDisc()displaylayer.drawDisc()****YDisplayLayer**

Dessine un disque plein à une position spécifiée.

```
function drawDisc( x, y, r)
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque

**y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque

**r** le rayon du disque, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawImage()displaylayer.drawImage()****YDisplayLayer**

Dessine une image GIF à la position spécifiée de la couche.

```
function drawImage( x, y, imagename)
```

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image

**y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image

**imagename** le nom du fichier GIF à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawPixel()  
displaylayer.drawPixel()****YDisplayLayer**

Dessine un pixel unique à une position spécifiée.

```
function drawPixel( x, y )
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche

**y** la distance en pixels depuis le haut de la couche

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawRect()displaylayer.drawRect()****YDisplayLayer**

Dessine un rectangle vide à une position spécifiée.

```
function drawRect( x1, y1, x2, y2)
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

**y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

**x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

**y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawText()displaylayer.drawText()****YDisplayLayer**

Affiche un texte à la position spécifiée de la couche.

```
function drawText( x, y, anchor, text)
```

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte

**y** la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte

**anchor** le point d'ancrage du texte, choisi parmi l'énumération Y\_ALIGN: Y\_ALIGN\_TOP\_LEFT,  
Y\_ALIGN\_CENTER\_LEFT, Y\_ALIGN\_BASELINE\_LEFT, Y\_ALIGN\_BOTTOM\_LEFT,  
Y\_ALIGN\_TOP\_CENTER, Y\_ALIGN\_CENTER, Y\_ALIGN\_BASELINE\_CENTER,  
Y\_ALIGN\_BOTTOM\_CENTER, Y\_ALIGN\_TOP\_DECIMAL,  
Y\_ALIGN\_CENTER\_DECIMAL, Y\_ALIGN\_BASELINE\_DECIMAL,  
Y\_ALIGN\_BOTTOM\_DECIMAL, Y\_ALIGN\_TOP\_RIGHT, Y\_ALIGN\_CENTER\_RIGHT,  
Y\_ALIGN\_BASELINE\_RIGHT, Y\_ALIGN\_BOTTOM\_RIGHT.

**text** le texte à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→get\_display()**

**YDisplayLayer**

**displaylayer→display()displaylayer.get\_display()**

---

Retourne l'YDisplay parent.

```
function 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.

```
function 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.

```
function 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.

```
function 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.

**function get\_layerWidth( )**

**Retourne :**

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERWIDTH\_INVALID.

**displaylayer→hide()displaylayer.hide()****YDisplayLayer**

Cache la couche de dessin.

```
function 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.

```
function lineTo( x, y)
```

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au point final

**y** la distance en pixels depuis le haut de la couche jusqu'au point final

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→moveTo()displaylayer.moveTo()****YDisplayLayer**

Déplace le point de dessin courant de cette couche à la position spécifiée.

function **moveTo( x, y )**

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche de dessin

**y** la distance en pixels depuis le haut de la couche de dessin

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→reset()displaylayer.reset()****YDisplayLayer**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

**function 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.

function **selectColorPen( color)**

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

**Paramètres :**

**color** la couleur RGB désirée (sous forme d'entier 24 bits)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectEraser()  
displaylayer.selectEraser()**

**YDisplayLayer**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

**function 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.

```
function selectFont( fontname )
```

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

**Paramètres :**

**fontname** le nom du fichier définissant la police de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectGrayPen()**  
**displaylayer.selectGrayPen()****YDisplayLayer**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

**function selectGrayPen( graylevel)**

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), 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.

function **setAntialiasingMode( 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()**  
**displaylayer.setConsoleBackground()****YDisplayLayer**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

```
function setConsoleBackground( bgcol)
```

**Paramètres :**

**bgcol** le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleMargins()**  
**displaylayer.setConsoleMargins()****YDisplayLayer**

Configure les marges d'affichage pour la fonction `consoleOut`.

```
function setConsoleMargins( x1, y1, x2, y2 )
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche

**y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure

**x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite

**y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## displaylayer→setConsoleWordWrap() displaylayer.setConsoleWordWrap()

---

YDisplayLayer

Configure le mode de retour à la ligne utilisé par la fonction consoleOut.

```
function setConsoleWordWrap( 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()**  
**displaylayer.setLayerPosition()****YDisplayLayer**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

```
function setLayerPosition( x, y, scrollTime)
```

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

**Paramètres :**

- x** la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
- y** la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
- scrollTime** durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## displaylayer→unhide()**displaylayer.unhide()**

YDisplayLayer

Affiche la couche.

**function 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)**

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.

```
function yFindDualPower( 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.

```
function 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()dualpower.describe()****YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 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).

```
function 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.

```
function 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()**

**YDualPower**

**dualpower→errorType()dualpower.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

**dualpower→get\_extVoltage()****YDualPower****dualpower→extVoltage()dualpower.get\_extVoltage()**

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

```
function 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.

```
function 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.

**function 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()**

**YDualPower**

**dualpower→functionId()dualpower.get\_functionId()**

---

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

**function get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**dualpower→get\_hardwareId()****YDualPower****dualpower→hardwareId()dualpower.get\_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'alimentation au format SERIAL.FUNCTIONID.

```
function 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.

```
function 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.

```
function 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\_module\_async()**  
**dualpower→module\_async()**  
**dualpower.get\_module\_async()****YDualPower**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**dualpower→get\_powerControl()**  
**dualpower→powerControl()**  
**dualpower.get\_powerControl()**

**YDualPower**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
function 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()**  
**dualpower→powerState()**  
**dualpower.get\_powerState()**

**YDualPower**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

**function get\_powerState( )**

**Retourne :**

une valeur parmi Y\_POWERSTATE\_OFF, Y\_POWERSTATE\_FROM\_USB et Y\_POWERSTATE\_FROM\_EXT représentant la source d'alimentation active pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y\_POWERSTATE\_INVALID.

**dualpower→get(userData)****YDualPower****dualpower→userData()dualpower.get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **dualpower→isOnline()dualpower.isOnline()**

**YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

**function 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→isOnline\_async()**  
**dualpower.isOnline\_async()****YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**dualpower→load()dualpower.load()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→load\_async()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**dualpower→nextDualPower()**  
**dualpower.nextDualPower()**

---

**YDualPower**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower( )`.

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→set\_powerControl()**  
**dualpower→setPowerControl()**  
**dualpower.set\_powerControl()**

**YDualPower**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

function **set\_powerControl( newval )**

**Paramètres :**

**newval** une valeur parmi Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT et Y\_POWERCONTROL\_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→set(userData)**

**YDualPower**

**dualpower→setUserData()dualpower.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**dualpower→wait\_async()dualpower.wait\_async()****YDualPower**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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() yFindFiles()yFindFiles()

YFiles

Permet de retrouver un système de fichier d'après un identifiant donné.

```
function yFindFiles( 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() yFirstFiles()yFirstFiles()

YFiles

Commence l'énumération des système de fichier accessibles par la librairie.

function **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(files.describe())****YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE (NAME )=SERIAL.FUNCTIONID.

**function 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()files.download()**

YFiles

Télécharge le fichier choisi du filesystem et retourne son contenu.

```
function download( pathname)
```

**Paramètres :**

**pathname** nom complet du fichier à charger, y compris le chemin d'accès.

**Retourne :**

le contenu du fichier chargé sous forme d'objet binaire

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

**files→download\_async()files.download\_async()****YFiles**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
function download_async( pathname, callback, context)
```

**Paramètres :**

**pathname** nom complet du fichier à charger, y compris le chemin d'accès.

**callback** fonction fournie par l'utilisateur, qui sera appelée lorsque la suite du chargement aura été effectué. La fonction callback doit prendre trois arguments: - la variable de contexte à disposition de l'utilisateur - l'objet YFiles dont la méthode download\_async a été appelée - le contenu du fichier chargé sous forme d'objet binaire

**context** variable de contexte à disposition de l'utilisateur

**Retourne :**

rien.

**files→format\_fs()files.format\_fs()****YFiles**

Rétabli le système de fichier dans un état original, défragmenté.

**function 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.

<b>files→get_advertisedValue()</b>	<b>YFiles</b>
<b>files→advertisedValue()files.get_advertisedValue()</b>	

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**files→getErrorMessage()****YFiles****files→errorMessage()files.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
function 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.

**function 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.

```
function 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.

**function get\_freeSpace( )**

**Retourne :**

un entier représentant l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets

En cas d'erreur, déclenche une exception ou retourne Y\_FREESPACE\_INVALID.

**files→get\_friendlyName()****YFiles****files→friendlyName()files.get\_friendlyName()**

Retourne un identifiant global du système de fichier au format NOM\_MODULE . NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**files→get\_functionId()****YFiles****files→functionId()files.get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

```
function 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()**  
**files→hardwareId(files.get\_hardwareId())****YFiles**

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: RELAYL01-123456.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.

```
function get_list( 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.

<b>files→get_logicalName()</b>	<b>YFiles</b>
<b>files→logicalName()files.get_logicalName()</b>	

Retourne le nom logique du système de fichier.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du système de fichier. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**files→get\_module()****YFiles****files→module()files.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function 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>files→get_module_async()</b>	<b>YFiles</b>
<b>files→module_async()files.get_module_async()</b>	

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**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).

```
function 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()files.isOnline()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()files.isOnline\_async()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**files→load()files.load()****YFiles**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→load\_async()files.load\_async()****YFiles**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## files→nextFiles()files.nextFiles()

YFiles

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles()`.

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()files.remove()****YFiles**

Efface un fichier, spécifié par son path complet, du système de fichier.

**function remove( pathname)**

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occupe. Par contre, la ré-écriture d'un fichier du même nom récupérera dans tout les cas l'espace qui n'aurait éventuellement pas été libéré. Pour s'assurer de libérer la totalité de l'espace du système de fichier, utilisez la fonction `format_fs`.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→set\_logicalName()**  
**files→setLogicalName()files.set\_logicalName()****YFiles**

Modifie le nom logique du système de fichier.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du système de fichier.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→set(userData)** YFiles  
**files→setUserData(files.set(userData))**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**files→upload()files.upload()**

YFiles

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

```
function upload( pathname, content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→wait\_async()files.wait\_async()****YFiles**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.17. Interface de la fonction GenericSensor

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_genericsensor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGenericSensor = yoctolib.YGenericSensor;
php require_once('yocto_genericsensor.php');
cpp #include "yocto_genericsensor.h"
m #import "yocto_genericsensor.h"
pas uses yocto_genericsensor;
vb yocto_genericsensor.vb
cs yocto_genericsensor.cs
java import com.yoctopuce.YoctoAPI.YGenericSensor;
py from yocto_genericsensor import *

```

### Fonction globales

#### **yFindGenericSensor(func)**

Permet de retrouver un capteur générique d'après un identifiant donné.

#### **yFirstGenericSensor()**

Commence l'énumération des capteurs génériques accessibles par la librairie.

### Méthodes des objets **YGenericSensor**

#### **genericsensor→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **genericsensor→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **genericsensor→get\_advertisedValue()**

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

#### **genericsensor→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **genericsensor→get\_currentValue()**

Retourne la valeur mesurée actuelle.

#### **genericsensor→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### **genericsensor→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### **genericsensor→get\_friendlyName()**

Retourne un identifiant global du capteur générique au format NOM\_MODULE . NOM\_FONCTION.

#### **genericsensor→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **genericsensor→get\_functionId()**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

#### **genericsensor→get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur générique au format SERIAL . FUNCTIONID.
<b>genericsensor→get_highestValue()</b>
Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.
<b>genericsensor→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>genericsensor→get_logicalName()</b>
Retourne le nom logique du capteur générique.
<b>genericsensor→get_lowestValue()</b>
Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.
<b>genericsensor→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>genericsensor→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>genericsensor→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>genericsensor→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>genericsensor→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>genericsensor→get_signalRange()</b>
Retourne la plage de signal électrique utilisée par le capteur.
<b>genericsensor→get_signalUnit()</b>
Retourne l'unité du signal électrique utilisée par le capteur.
<b>genericsensor→get_signalValue()</b>
Retourne la valeur mesurée du signal électrique utilisée par le capteur.
<b>genericsensor→get_unit()</b>
Retourne l'unité dans laquelle la mesure est exprimée.
<b>genericsensor→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>genericsensor→get_valueRange()</b>
Retourne la plage de valeurs physiques mesurés par le capteur.
<b>genericsensor→isOnline()</b>
Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.
<b>genericsensor→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.
<b>genericsensor→load(msValidity)</b>
Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.
<b>genericsensor→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>genericsensor→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.
<b>genericsensor→nextGenericSensor()</b>

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.

```
function yFindGenericSensor( 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.

```
function 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()**  
**genericsensor.calibrateFromPoints()****YGenericSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**function calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**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.

function **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 générique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**genericsensor→get\_advertisedValue()**  
**genericsensor→advertisedValue()**  
**genericsensor.get\_advertisedValue()**

---

**YGenericSensor**

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

**function 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).

```
function 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.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**genericsensor→getErrorMessage()**  
**genericsensor→errorMessage()**  
**genericsensor.getErrorMessage()**

**YGenericSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

```
function 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 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.

**function 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()**  
**genericsensor→friendlyName()**  
**genericsensor.get\_friendlyName()**

**YGenericSensor**

Retourne un identifiant global du capteur générique au format NOM\_MODULE . NOM\_FONCTION.

**function 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()**  
**genericsensor→functionDescriptor()**  
**genericsensor.get\_functionDescriptor()**

**YGenericSensor**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**genericsensor→get\_functionId()**  
**genericsensor→functionId()**  
**genericsensor.get\_functionId()**

**YGenericSensor**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

**function get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**genericsensor→get\_hardwareId()**  
**genericsensor→hardwareId()**  
**genericsensor.get\_hardwareId()**

**YGenericSensor**

Retourne l'identifiant matériel unique du capteur générique au format SERIAL.FUNCTIONID.

**function 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()**  
**genericsensor→highestValue()**  
**genericsensor.get\_highestValue()**

**YGenericSensor**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

```
function get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**genericsensor→get\_logFrequency()**  
**genericsensor→logFrequency()**  
**genericsensor.get\_logFrequency()**

**YGenericSensor**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function 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()**  
**genericsensor→logicalName()**  
**genericsensor.get\_logicalName()**

**YGenericSensor**

Retourne le nom logique du capteur générique.

```
function 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.

**function 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.

**function 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\_module\_async()**  
**genericsensor→module\_async()**  
**genericsensor.get\_module\_async()**

**YGenericSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**genericsensor→get\_recordedData()**  
**genericsensor→recordedData()**  
**genericsensor.get\_recordedData()**

**YGenericSensor**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( startTime, endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**genericsensor→get\_reportFrequency()**  
**genericsensor→reportFrequency()**  
**genericsensor.get\_reportFrequency()**

**YGenericSensor**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **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.

**function 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()**  
**genericsensor→signalRange()**  
**genericsensor.get\_signalRange()**

---

**YGenericSensor**

Retourne la plage de signal électrique utilisée par le capteur.

**function 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.

```
function 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()**  
**genericsensor→signalValue()**  
**genericsensor.get\_signalValue()**

**YGenericSensor**

---

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

function **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.

function **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**genericsensor→get(userData)**  
**genericsensor→userData()**  
**genericsensor.get(userData)**

**YGenericSensor**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function 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.

```
function 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.

```
function 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→isOnline\_async()**  
**genericsensor.isOnline\_async()****YGenericSensor**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**genericsensor→load()genericsensor.load()****YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→loadCalibrationPoints()  
genericsensor.loadCalibrationPoints()****YGenericSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→load\_async()  
genericsensor.load\_async()****YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**genericsensor→nextGenericSensor()**  
**genericsensor.nextGenericSensor()****YGenericSensor**

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

```
function nextGenericSensor( )
```

**Retourne :**

un pointeur sur un objet `YGenericSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**genericsensor→registerTimedReportCallback()  
genericsensor.registerTimedReportCallback()****YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

**genericsensor→registerValueCallback()**  
**genericsensor.registerValueCallback()****YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**genericsensor→setHighestValue()**  
**genericsensor.set\_highestValue()**

**YGenericSensor**

Modifie la mémoire de valeur maximale observée.

function **set\_highestValue( newval )**

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_logFrequency()**  
**genericsensor→setLogFrequency()**  
**genericsensor.set\_logFrequency()**

**YGenericSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_logicalName()**  
**genericsensor→setLogicalName()**  
**genericsensor.set\_logicalName()**

**YGenericSensor**

Modifie le nom logique du capteur générique.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur générique.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_lowestValue()**  
**genericsensor→setLowestValue()**  
**genericsensor.set\_lowestValue()**

**YGenericSensor**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_reportFrequency()**  
**genericsensor→setReportFrequency()**  
**genericsensor.set\_reportFrequency()**

**YGenericSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_resolution()**  
**genericsensor→setResolution()**  
**genericsensor.set\_resolution()**

**YGenericSensor**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_signalRange()**  
**genericsensor→setSignalRange()**  
**genericsensor.set\_signalRange()**

**YGenericSensor**

Modifie la plage de signal électrique utilisée par le capteur.

function **set\_signalRange( newval)**

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_unit()****YGenericSensor****genericsensor→setUnit()genericsensor.set\_unit()**

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
function set_unit( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set(userData)**  
**genericsensor→setUserData()**  
**genericsensor.set(userData)**

**YGenericSensor**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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.

function **set\_valueRange( newval)**

Le changement de plage peut avoir pour effet de bord un changement automatique de la résolution affichée.

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→wait\_async()  
genericsensor.wait\_async()****YGenericSensor**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.18. Interface de la fonction Gyro

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_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 retournée par le capteur (sans arrondi ni calibration).

#### **gyro→get\_currentValue()**

Retourne la valeur actuelle de la vitesse angulaire.

#### **gyro→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

#### **gyro→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

#### **gyro→get\_friendlyName()**

Retourne un identifiant global du gyroscope au format NOM\_MODULE . NOM\_FONCTION.

#### **gyro→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **gyro→get\_functionId()**

Retourne l'identifiant matériel du gyroscope, sans référence au module.

#### **gyro→get\_hardwareId()**

Retourne l'identifiant matériel unique du gyroscope au format SERIAL . FUNCTIONID.

**gyro→get\_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

**gyro→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**gyro→get\_logicalName()**

Retourne le nom logique du gyroscope.

**gyro→get\_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

**gyro→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**gyro→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**gyro→get\_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionX()**

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionY()**

Retourne la composante y du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_quaternionZ()**

Retourne la composante z du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**gyro→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**gyro→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**gyro→get\_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**gyro→get\_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

**gyro→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**gyro→get\_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

#### **gyro→get\_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

#### **gyro→get\_zValue()**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

#### **gyro→isOnline()**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

#### **gyro→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

#### **gyro→load(msValidity)**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

#### **gyro→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **gyro→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

#### **gyro→nextGyro()**

Continue l'énumération des gyroscopes commencée à l'aide de yFirstGyro( ).

#### **gyro→registerAnglesCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

#### **gyro→registerQuaternionCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

#### **gyro→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### **gyro→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **gyro→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

#### **gyro→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### **gyro→set\_logicalName(newval)**

Modifie le nom logique du gyroscope.

#### **gyro→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

#### **gyro→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **gyro→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **gyro→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **gyro→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YGyro.FindGyro() yFindGyro()yFindGyro()

YGyro

Permet de retrouver un gyroscope d'après un identifiant donné.

**function yFindGyro( func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le gyroscope sans ambiguïté

### Retourne :

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

## YGyro.FirstGyro() yFirstGyro()yFirstGyro()

YGyro

Commence l'énumération des gyroscopes accessibles par la librairie.

function **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.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→describe()gyro.describe()****YGyro**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function 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 gyroscope (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**gyro→get\_advertisedValue()**

**YGyro**

**gyro→advertisedValue()gyro.get\_advertisedValue()**

---

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du gyroscope (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**gyro→get\_currentRawValue()****YGyro****gyro→currentRawValue()gyro.get\_currentRawValue()**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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()**  
**gyro→currentValue()gyro.get\_currentValue()**

---

**YGyro**

Retourne la valeur actuelle de la vitesse angulaire.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la vitesse angulaire

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

**gyro→get\_errorMessage()****YGyro****gyro→errorMessage()gyro.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
function 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()**  
**gyro→errorType()gyro.get\_errorType()**

---

**YGyro**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

**function 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.

```
function 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.

<b>gyro→get_functionDescriptor()</b>	<b>YGyro</b>
<b>gyro→functionDescriptor()</b>	
<b>gyro.get_functionDescriptor()</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**gyro→get\_functionId()****YGyro****gyro→functionId()gyro.get\_functionId()**

Retourne l'identifiant matériel du gyroscope, sans référence au module.

```
function 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`.

<b>gyro→get_hardwareId()</b>	<b>YGyro</b>
<b>gyro→hardwareId()gyro.get_hardwareId()</b>	

---

Retourne l'identifiant matériel unique du gyroscope au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**gyro→get\_heading()****YGyro****gyro→heading()gyro.get\_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function 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).

<b>gyro→get_highestValue()</b>	<b>YGyro</b>
<b>gyro→highestValue()gyro.get_highestValue()</b>	

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

```
function 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.

```
function 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.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du gyroscope. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**gyro→get\_lowestValue()****YGyro****gyro→lowestValue()gyro.get\_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

```
function 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**gyro→get\_module\_async()****YGyro****gyro→module\_async()gyro.get\_module\_async()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

<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.

**function 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.

```
function 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.

```
function 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.

```
function 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.

```
function 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.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

<b>gyro→get_reportFrequency()</b>	<b>YGyro</b>
<b>gyro→reportFrequency()gyro.get_reportFrequency()</b>	

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function 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.

```
function 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.

<b>gyro→get_roll()</b>	<b>YGyro</b>
<b>gyro→roll()gyro.get_roll()</b>	

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.

**function 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.

```
function 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).

```
function 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.

```
function get_xValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_XVALUE\_INVALID.

**gyro→get\_yValue()**

**YGyro**

**gyro→yValue()gyro.get\_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

```
function 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.

```
function get_zValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_ZVALUE\_INVALID.

**gyro→isOnline()gyro.isOnline()****YGyro**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()gyro.isOnline\_async()

YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**gyro→load()gyro.load()****YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→loadCalibrationPoints()  
gyro.loadCalibrationPoints()****YGyro**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→load\_async()gyro.load\_async()****YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**gyro→nextGyro()gyro.nextGyro()****YGyro**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

```
function nextGyro( )
```

**Retourne :**

un pointeur sur un objet `YGyro` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**gyro→registerAnglesCallback()  
gyro.registerAnglesCallback()****YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

**function registerAnglesCallback( callback)**

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas 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 quatre arguments: l'objet YGyro du module qui a tourné, et les valeurs des trois angles roll, pitch et heading en degrés (nombres à virgules).

**gyro→registerQuaternionCallback()  
gyro.registerQuaternionCallback()****YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
function registerQuaternionCallback( callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appellés 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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>gyro→set_highestValue()</b>	<b>YGyro</b>
<b>gyro→setHighestValue()gyro.set_highestValue()</b>	

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_logFrequency()****YGyro****gyro→setLogFrequency()gyro.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>gyro→set_logicalName()</b>	<b>YGyro</b>
<b>gyro→setLogicalName()gyro.set_logicalName()</b>	

---

Modifie le nom logique du gyroscope.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du gyroscope.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**gyro→set\_lowestValue()****YGyro****gyro→setLowestValue()gyro.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_reportFrequency()**  
**gyro→setReportFrequency()**  
**gyro.set\_reportFrequency()**

**YGyro**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_resolution()****YGyro****gyro→setResolution()gyro.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval )
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set(userData)**  
**gyro→setUserData()gyro.set(userData)**

**YGyro**

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).

```
function set(userData) {
```

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

**gyro→wait\_async()gyro.wait\_async()****YGyro**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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;
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)**

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() yFindHubPort()yFindHubPort()

**YHubPort**

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

**function yFindHubPort( 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.

```
function 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()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.

```
function 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 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).

```
function 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.

**function 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.

```
function 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→getErrorMessage()**

**YHubPort**

**hubport→errorMessage()hubport.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
function 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 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.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

## **hubport→get\_friendlyName()**

**YHubPort**

## **hubport→friendlyName()hubport.get\_friendlyName()**

Retourne un identifiant global du port de Yocto-hub au format NOM\_MODULE . NOM\_FONCTION.

```
function 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.

**function 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

**hubport→get\_functionId()**

**YHubPort**

**hubport→functionId()hubport.get\_functionId()**

---

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

**function get\_functionId( )**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**hubport→get\_hardwareId()****YHubPort****hubport→hardwareId()hubport.get\_hardwareId()**

Retourne l'identifiant matériel unique du port de Yocto-hub au format SERIAL.FUNCTIONID.

```
function 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()**

**YHubPort**

**hubport→logicalName()hubport.get\_logicalName()**

---

Retourne le nom logique du port de Yocto-hub.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port de Yocto-hub. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**hubport→get\_module()****YHubPort****hubport→module()hubport.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**hubport→get\_module\_async()**  
**hubport→module\_async()**  
**hubport.get\_module\_async()**

**YHubPort**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module\_async( callback, context)**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**hubport→get\_portState()****YHubPort****hubport→portState()hubport.get\_portState()**

Retourne l'état actuel du port de Yocto-hub.

```
function 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).

```
function 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.

**function 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→isOnline\_async()hubport.isOnline\_async()****YHubPort**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**hubport→load()hubport.load()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport→load\_async()hubport.load\_async()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**hubport→nextHubPort()hubport.nextHubPort()****YHubPort**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

```
function nextHubPort( )
```

**Retourne :**

un pointeur sur un objet `YHubPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**hubport→registerValueCallback()**  
**hubport.registerValueCallback()****YHubPort**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**hubport→setEnabled()hubport.set\_enabled()****YHubPort**

Modifie le mode d'activation du port du Yocto-hub.

```
function set_enabled( newval )
```

Si le port est actif, il sera alimenté. Sinon, l'alimentation du module est coupée.

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon le mode d'activation du port du Yocto-hub

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport→set\_logicalName()**  
**hubport→setLogicalName()**  
**hubport.set\_logicalName()**

**YHubPort**

Modifie le nom logique du port de Yocto-hub.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du port de Yocto-hub.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport→set(userData)****YHubPort****hubport→setUserData()hubport.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**hubport→wait\_async()hubport.wait\_async()****YHubPort**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.20. Interface de la fonction Humidity

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_humidity.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YHumidity = yoctolib.YHumidity;
php	require_once('yocto_humidity.php');
cpp	#include "yocto_humidity.h"
m	#import "yocto_humidity.h"
pas	uses yocto_humidity;
vb	yocto_humidity.vb
cs	yocto_humidity.cs
java	import com.yoctopuce.YoctoAPI.YHumidity;
py	from yocto_humidity import *

### Fonction globales

#### **yFindHumidity(func)**

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

#### **yFirstHumidity()**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

### Méthodes des objets YHumidity

#### **humidity→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **humidity→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **humidity→get\_advertisedValue()**

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

#### **humidity→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **humidity→get\_currentValue()**

Retourne la mesure actuelle de l'humidité.

#### **humidity→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### **humidity→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### **humidity→get\_friendlyName()**

Retourne un identifiant global du capteur d'humidité au format NOM\_MODULE . NOM\_FONCTION.

#### **humidity→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **humidity→get\_functionId()**

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

#### **humidity→get\_hardwareId()**

### 3. Reference

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>

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**humidity→set\_logicalName(newval)**

Modifie le nom logique du capteur d'humidité.

**humidity→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour l'humidité.

**humidity→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**humidity→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**humidity→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**humidity→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YHumidity.FindHumidity() yFindHumidity()yFindHumidity()

YHumidity

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

```
function yFindHumidity( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur d'humidité soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnline()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

### Retourne :

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

**YHumidity.FirstHumidity()****yFirstHumidity()yFirstHumidity()****YHumidity**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

```
function 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()****YHumidity****humidity.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**function calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→describe()humidity.describe()****YHumidity**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 d'humidité (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**humidity→get\_advertisedValue()**

**YHumidity**

**humidity→advertisedValue()**

**humidity.get\_advertisedValue()**

---

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

```
function 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()**  
**humidity→currentRawValue()**  
**humidity.get\_currentRawValue()**

**YHumidity**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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()**

**YHumidity**

**humidity→currentValue()humidity.get\_currentValue()**

---

Retourne la mesure actuelle de l'humidité.

```
function 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()**  
**humidity→errorMessage()**  
**humidity.getErrorMessage()****YHumidity**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
function 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.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

**humidity→get\_friendlyName()**  
**humidity→friendlyName()**  
**humidity.get\_friendlyName()**

**YHumidity**

Retourne un identifiant global du capteur d'humidité au format NOM\_MODULE . NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**humidity→get\_functionId()****YHumidity****humidity→functionId()humidity.get\_functionId()**

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

```
function 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.

```
function 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.

```
function 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.

```
function 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.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur d'humidité. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**humidity→get\_lowestValue()**

**YHumidity**

**humidity→lowestValue()humidity.get\_lowestValue()**

---

Retourne la valeur minimale observée pour l'humidité.

```
function 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.

```
function 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\_module\_async()**  
**humidity→module\_async()**  
**humidity.get\_module\_async()****YHumidity**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**humidity→get\_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.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**humidity→get\_reportFrequency()**  
**humidity→reportFrequency()**  
**humidity.get\_reportFrequency()**

**YHumidity**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**function get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**humidity→get\_resolution()****YHumidity****humidity→resolution()humidity.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function 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.

```
function 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).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**humidity→isOnline()**humidity.isOnline()******YHumidity**

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

**function 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→isOnline\_async()**  
**humidity.isOnline\_async()****YHumidity**

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**humidity→load()**humidity.load()******YHumidity**

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→loadCalibrationPoints()****YHumidity****humidity.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→load\_async()humidity.load\_async()****YHumidity**

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**humidity→nextHumidity()humidity.nextHumidity()****YHumidity**

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

```
function nextHumidity( )
```

**Retourne :**

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**humidity→registerTimedReportCallback()  
humidity.registerTimedReportCallback()****YHumidity**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

**humidity→registerValueCallback()**  
**humidity.registerValueCallback()****YHumidity**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**humidity→setHighestValue()**  
**humidity.set\_highestValue()**

YHumidity

Modifie la mémoire de valeur maximale observée pour l'humidité.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_logFrequency()**  
**humidity→setLogFrequency()**  
**humidity.set\_logFrequency()****YHumidity**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_logicalName()**  
**humidity→setLogicalName()**  
**humidity.set\_logicalName()****YHumidity**

Modifie le nom logique du capteur d'humidité.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur d'humidité.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_lowestValue()**  
**humidity→setLowestValue()**  
**humidity.set\_lowestValue()**

**YHumidity**

Modifie la mémoire de valeur minimale observée pour l'humidité.

```
function set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>humidity→set_reportFrequency()</b>	<b>YHumidity</b>
<b>humidity→setReportFrequency()</b>	
<b>humidity.set_reportFrequency()</b>	

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_resolution()****YHumidity****humidity→setResolution()humidity.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval )
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## humidity→set(userData)

YHumidity

## humidity→setUserData()humidity.set(userData)

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**humidity→wait\_async()humidity.wait\_async()****YHumidity**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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.

**led→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**led→get\_power()**

Retourne l'état courant de la led.

**led→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**led→isOnline()**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

**led→isOnline\_async(callback, context)**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

**led→load(msValidity)**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

**led→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

**led→nextLed()**

Continue l'énumération des leds commencée à l'aide de yFirstLed( ).

**led→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**led→set\_blinking(newval)**

Modifie le mode de signalisation de la led.

**led→set\_logicalName(newval)**

Modifie le nom logique de la led.

**led→set\_luminosity(newval)**

Modifie l'intensité lumineuse de la led (en pour cent).

**led→set\_power(newval)**

Modifie l'état courant de la led.

**led→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**led→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YLed.FindLed() yFindLed()yFindLed()

YLed

Permet de retrouver une led d'après un identifiant donné.

```
function yFindLed( 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.

```
function 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()led.describe()**

YLed

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**function 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 la led (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

---

**led→get\_advertisedValue()****YLed****led→advertisedValue()led.get\_advertisedValue()**

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
function 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.

```
function 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→getErrorMessage()****YLed****led→errorMessage()led.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

```
function 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.

**led→get\_errorType()**  
**led→errorType()led.get\_errorType()**

YLed

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

**function 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.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**led→get\_functionId()****YLed****led→functionId()led.get\_functionId()**

Retourne l'identifiant matériel de la led, sans référence au module.

```
function 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.

```
function 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.

```
function 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.

```
function 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.

```
function 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\_module\_async()**

YLed

**led→module\_async()led.get\_module\_async()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**led→get\_power()****YLed****led→power()led.get\_power()**

Retourne l'état courant de la led.

```
function 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).

```
function 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.

```
function 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→isOnline\_async()|led.isOnline\_async()**

YLed

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**led→load()led.load()****YLed**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→load\_async()|led.load\_async()**

YLed

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**led→nextLed()led.nextLed()****YLed**

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_blinking( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led->set\_logicalName()** YLed  
**led->setLogicalName()|led.set\_logicalName()**

Modifie le nom logique de la led.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la led.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→set\_luminosity()****YLed****led→setLuminosity()led.set\_luminosity()**

Modifie l'intensité lumineuse de la led (en pour cent).

```
function set_luminosity( newval)
```

**Paramètres :**

**newval** un entier représentant l'intensité lumineuse de la led (en pour cent)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led->set\_power()**  
**led->setPower()led.set\_power()**

---

YLed

Modifie l'état courant de la led.

```
function set_power( newval)
```

**Paramètres :**

**newval** soit Y\_POWER\_OFF, soit Y\_POWER\_ON, selon l'état courant de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→set(userData)****YLed****led→setUserData()|led.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**led→wait\_async()|led.wait\_async()**

YLed

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.22. Interface de la fonction LightSensor

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_lightsensor.js'></script>
node.js	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 renvoyée 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>

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

**lightsensor→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**lightsensor→set\_logicalName(newval)**

Modifie le nom logique du capteur de lumière.

**lightsensor→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

**lightsensor→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**lightsensor→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**lightsensor→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**lightsensor→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YLightSensor.FindLightSensor() yFindLightSensor()yFindLightSensor()

YLightSensor

Permet de retrouver un capteur de lumière d'après un identifiant donné.

```
function yFindLightSensor( 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.

```
function 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()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).

```
function calibrate( calibratedVal)
```

**Paramètres :**

**calibratedVal** la consigne de valeur désirée.

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→calibrateFromPoints()**  
**lightsensor.calibrateFromPoints()****YLightSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→describe()lightsensor.describe()****YLightSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format TYPE (NAME )=SERIAL.FUNCTIONID.

**function 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 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).

```
function 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()**  
**lightsensor→currentRawValue()**  
**lightsensor.get\_currentRawValue()**

**YLightSensor**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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.

```
function 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→getErrorMessage()**  
**lightsensor→errorMessage()**  
**lightsensor.getErrorMessage()**

**YLightSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

**function 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 lumière.

**lightsensor→get\_errorType()****YLightSensor****lightsensor→errorType()lightsensor.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
function 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()**  
**lightsensor→friendlyName()**  
**lightsensor.get\_friendlyName()**

**YLightSensor**

Retourne un identifiant global du capteur de lumière au format NOM\_MODULE . NOM\_FONCTION.

**function 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.

**lightsensor→get\_functionDescriptor()****YLightSensor****lightsensor→functionDescriptor()****lightsensor.get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function 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

**lightsensor→get\_functionId()**

**YLightSensor**

**lightsensor→functionId()lightsensor.get\_functionId()**

---

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

**function 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\_hwId()  
lightsensor→hwId()  
lightsensor.get\_hwId()****YLightSensor**

Retourne l'identifiant matériel unique du capteur de lumière au format SERIAL.FUNCTIONID.

```
function get_hwId( )
```

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()**  
**lightsensor→highestValue()**  
**lightsensor.get\_highestValue()**

**YLightSensor**

Retourne la valeur maximale observée pour la lumière ambiante.

**function 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()**  
**lightsensor→logFrequency()**  
**lightsensor.get\_logFrequency()**

**YLightSensor**

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.

function **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.

```
function 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.

```
function get_lowestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**lightsensor→get\_module()**

**YLightSensor**

**lightsensor→module()lightsensor.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**lightsensor→get\_module\_async()**  
**lightsensor→module\_async()**  
**lightsensor.get\_module\_async()****YLightSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**lightsensor→get\_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.

**function get\_recordedData( startTime, endTime )**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**lightsensor→get\_reportFrequency()**  
**lightsensor→reportFrequency()**  
**lightsensor.get\_reportFrequency()**

**YLightSensor**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**lightsensor→get\_resolution()**

**YLightSensor**

**lightsensor→resolution()lightsensor.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
function 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.

```
function get_unit( )
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la lumière ambiante est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**lightsensor→get(userData)**

**YLightSensor**

**lightsensor→userData()lightsensor.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**lightsensor→isOnline()lightsensor.isOnline()****YLightSensor**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

**function 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→isOnline\_async()**  
**lightsensor.isOnline\_async()****YLightSensor**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**lightsensor→load()lightsensor.load()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

function **load( msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→loadCalibrationPoints()**  
**lightsensor.loadCalibrationPoints()****YLightSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→load\_async()lightsensor.load\_async()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**lightsensor→nextLightSensor()**  
**lightsensor.nextLightSensor()**

---

**YLightSensor**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

```
function nextLightSensor( )
```

**Retourne :**

un pointeur sur un objet `YLightSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

## lightsensor→registerTimedReportCallback() lightsensor.registerTimedReportCallback()

YLightSensor

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**lightsensor→registerValueCallback()**  
**lightsensor.registerValueCallback()****YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**lightsensor→setHighestValue()**  
**lightsensor.set\_highestValue()**

**YLightSensor**

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la lumière ambiante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logFrequency()**  
**lightsensor→setLogFrequency()**  
**lightsensor.set\_logFrequency()**

**YLightSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logicalName()**  
**lightsensor→setLogicalName()**  
**lightsensor.set\_logicalName()**

**YLightSensor**

Modifie le nom logique du capteur de lumière.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du capteur de lumière.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_lowestValue()**  
**lightsensor→setLowestValue()**  
**lightsensor.set\_lowestValue()**

**YLightSensor**

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

function **set\_lowestValue( newval )**

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la lumière ambiante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_reportFrequency()**  
**lightsensor→setReportFrequency()**  
**lightsensor.set\_reportFrequency()****YLightSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_resolution()**  
**lightsensor→setResolution()**  
**lightsensor.set\_resolution()**

**YLightSensor**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set(userData)**  
**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).

```
function set(userData) {
```

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

**lightsensor→wait\_async()lightsensor.wait\_async()****YLightSensor**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.23. Interface de la fonction Magnetometer

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_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 retournée par le capteur (sans arrondi ni calibration).

#### magnetometer→get\_currentValue()

Retourne la valeur actuelle du champ magnétique.

#### magnetometer→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### magnetometer→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### magnetometer→get\_friendlyName()

Retourne un identifiant global du magnétomètre au format NOM\_MODULE . NOM\_FONCTION.

#### magnetometer→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### magnetometer→get\_functionId()

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

#### magnetometer→get\_hardwareId()

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL . FUNCTIONID.

<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.

**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)**

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.

## Y Magnetometer.FindMagnetometer() yFindMagnetometer()yFindMagnetometer()

Y Magnetometer

Permet de retrouver un magnétomètre d'après un identifiant donné.

```
function yFindMagnetometer( 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 `Y Magnetometer.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 `Y Magnetometer` qui permet ensuite de contrôler le magnétomètre.

**YMagnetometer.FirstMagnetometer()****yFirstMagnetometer()yFirstMagnetometer()****YMagnetometer**

Commence l'énumération des magnétomètres accessibles par la librairie.

```
function 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()**  
**magnetometer.calibrateFromPoints()****YMagnetometer**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**function calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**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.

function **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 magnétomètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**magnetometer→get\_advertisedValue()**  
**magnetometer→advertisedValue()**  
**magnetometer.get\_advertisedValue()**

**YMagnetometer**

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

```
function 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()**  
**magnetometer→currentRawValue()**  
**magnetometer.get\_currentRawValue()**

**YMagnetometer**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**magnetometer→get\_currentValue()**  
**magnetometer→currentValue()**  
**magnetometer.get\_currentValue()**

---

**YMagnetometer**

Retourne la valeur actuelle du champ magnétique.

```
function 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→getErrorMessage()**  
**magnetometer→errorMessage()**  
**magnetometer.getErrorMessage()****YMagnetometer**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
function 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 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.

**function 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()**  
**magnetometer→friendlyName()**  
**magnetometer.get\_friendlyName()**

**YMagnetometer**

Retourne un identifiant global du magnétomètre au format NOM\_MODULE . NOM\_FONCTION.

```
function 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()**  
**magnetometer→functionDescriptor()**  
**magnetometer.get\_functionDescriptor()**

---

**YMagnetometer**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**magnetometer→get\_functionId()**  
**magnetometer→functionId()**  
**magnetometer.get\_functionId()**

**YMagnetometer**

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**magnetometer→get\_hardwareId()**  
**magnetometer→hardwareId()**  
**magnetometer.get\_hardwareId()**

**YMagnetometer**

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL.FUNCTIONID.

**function 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()**  
**magnetometer→highestValue()**  
**magnetometer.get\_highestValue()**

**YMagnetometer**

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

```
function get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**magnetometer→get\_logFrequency()**  
**magnetometer→logFrequency()**  
**magnetometer.get\_logFrequency()**

**YMagnetometer**

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.

**function 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()**  
**magnetometer→logicalName()**  
**magnetometer.get\_logicalName()**

**YMagnetometer**

Retourne le nom logique du magnétomètre.

```
function 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()**  
**magnetometer→lowestValue()**  
**magnetometer.get\_lowestValue()**

**YMagnetometer**

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

**function get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_LOWESTVALUE\_INVALID**.

**magnetometer→get\_module()**  
**magnetometer→module()**  
**magnetometer.get\_module()**

**YMagnetometer**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function 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\_module\_async()**  
**magnetometer→module\_async()**  
**magnetometer.get\_module\_async()****YMagnetometer**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**magnetometer→get\_recordedData()**  
**magnetometer→recordedData()**  
**magnetometer.get\_recordedData()**

**YMagnetometer**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( startTime, endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**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.

**function 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()**  
**magnetometer→resolution()**  
**magnetometer.get\_resolution()**

**YMagnetometer**

Retourne la résolution des valeurs mesurées.

**function 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.

```
function 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)**  
**magnetometer→userData()**  
**magnetometer.get(userData)**

**YMagnetometer**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function 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.

```
function 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.

```
function 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.

```
function 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()****YMagnetometer**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

**function 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→isOnline\_async()**  
**magnetometer.isOnline\_async()****YMagnetometer**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**magnetometer→load()****magnetometer.load()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

function **load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→loadCalibrationPoints()**  
**magnetometer.loadCalibrationPoints()****YMagnetometer**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→load\_async()**  
**magnetometer.load\_async()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**magnetometer→nextMagnetometer()**  
**magnetometer.nextMagnetometer()**

**YMagnetometer**

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

```
function nextMagnetometer( )
```

**Retourne :**

un pointeur sur un objet `YMagnetometer` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**magnetometer→registerTimedReportCallback()  
magnetometer.registerTimedReportCallback()****YMagnetometer**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**magnetometer→registerValueCallback()  
magnetometer.registerValueCallback()****YMagnetometer**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**magnetometer→setHighestValue()**  
**magnetometer.set\_highestValue()**

**YMagnetometer**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_logFrequency()**  
**magnetometer→setLogFrequency()**  
**magnetometer.set\_logFrequency()**

**YMagnetometer**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_logicalName()**  
**magnetometer→setLogicalName()**  
**magnetometer.set\_logicalName()**

**YMagnetometer**

Modifie le nom logique du magnétomètre.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du magnétomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_lowestValue()**  
**magnetometer→setLowestValue()**  
**magnetometer.set\_lowestValue()**

**YMagnetometer**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_reportFrequency()**  
**magnetometer→setReportFrequency()**  
**magnetometer.set\_reportFrequency()**

**YMagnetometer**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_resolution()**  
**magnetometer→setResolution()**  
**magnetometer.set\_resolution()**

**YMagnetometer**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set(userData)**  
**magnetometer→setUserData()**  
**magnetometer.set(userData)**

**YMagnetometer**

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).

```
function 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

**magnetometer→wait\_async()**  
**magnetometer.wait\_async()****YMagnetometer**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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.

**function get\_averageValue( )**

**Retourne :**

un nombre décimal correspondant à la valeur moyenne observée.

**measure→get\_endTimeUTC()****YMeasure****measure→endTimeUTC()measure.get\_endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

```
function 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.

function **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.

```
function get_minValue( )
```

**Retourne :**

un nombre décimal correspondant à la plus petite valeur observée.

<b>measure→getStartTimeUTC()</b>	<b>YMeasure</b>
<b>measure→startTimeUTC()</b>	
<b>measure.getStartTimeUTC()</b>	

---

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).

**function 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>
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

#### 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()

### 3. Reference

Retourne l'icône du module.

#### **module→get\_lastLogs()**

Retourne une chaîne de caractère contenant les derniers logs du module.

#### **module→get\_logicalName()**

Retourne le nom logique du module.

#### **module→get\_luminosity()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

#### **module→get\_persistentSettings()**

Retourne l'état courant des réglages persistents du module.

#### **module→get\_productId()**

Retourne l'identifiant USB du module, préprogrammé en usine.

#### **module→get\_productName()**

Retourne le nom commercial du module, préprogrammé en usine.

#### **module→get\_productRelease()**

Retourne le numéro de version matériel du module, préprogrammé en usine.

#### **module→get\_rebootCountdown()**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

#### **module→get\_serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

#### **module→get\_upTime()**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

#### **module→get\_usbBandwidth()**

Retourne le nombre d'interface USB utilisé par le module.

#### **module→get\_usbCurrent()**

Retourne le courant consommé par le module sur le bus USB, en milliampères.

#### **module→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **module→isOnline()**

Vérifie si le module est joignable, sans déclencher d'erreur.

#### **module→isOnline\_async(callback, context)**

Vérifie si le module est joignable, sans déclencher d'erreur.

#### **module→load(msValidity)**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

#### **module→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

#### **module→nextModule()**

Continue l'énumération des modules commencée à l'aide de yFirstModule( ).

#### **module→reboot(secBeforeReboot)**

Agende un simple redémarrage du module dans un nombre donné de secondes.

#### **module→registerLogCallback(callback)**

todo

#### **module→revertFromFlash()**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

#### **module→saveToFlash()**

Sauve les réglages courants dans la mémoire non volatile du module.

**module→set\_beacon(newval)**

Allume ou éteint la balise de localisation du module.

**module→set\_logicalName(newval)**

Change le nom logique du module.

**module→set\_luminosity(newval)**

Modifie la luminosité des leds informatives du module.

**module→set\_usbBandwidth(newval)**

Modifie le nombre d'interface USB utilisé par le module.

**module→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**module→triggerFirmwareUpdate(secBeforeReboot)**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

**module→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YModule.FindModule()  
yFindModule()yFindModule()****YModule**

Permet de retrouver un module d'après son numéro de série ou son nom logique.

**function yFindModule( 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.

```
function 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()module.describe()

YModule

Retourne un court texte décrivant le module.

**function 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.

```
function download( pathname)
```

**Paramètres :**

pathname nom complet du fichier

**Retourne :**

le contenu du fichier chargé

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

## module→functionCount()module.functionCount()

YModule

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

**function 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.

function **functionId( functionIndex)**

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

## module→functionName()module.functionName()

YModule

Retourne le nom logique de la *n*ième fonction du module.

**function** **functionName( functionIndex)**

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant au nom logique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module→functionValue()module.functionValue()****YModule**

Retourne la valeur publiée par la *nième* fonction du module.

function **functionValue( functionIndex)**

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module→get\_beacon()**  
**module→beacon()module.get\_beacon()**

---

**YModule**

Retourne l'état de la balise de localisation.

**function 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→getErrorMessage()****YModule****module→errorMessage()module.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
function 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 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.

**function 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.

```
function get_firmwareRelease( )
```

**Retourne :**

une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne Y\_FIRMWARERELEASE\_INVALID.

**module→get\_hwId()**

**YModule**

**module→hardwareId()module.get\_hwId()**

---

Retourne l'identifiant unique du module.

**function get\_hwId( )**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

**Retourne :**

une chaîne de caractères identifiant la fonction

---

**module→get\_icon2d()****YModule****module→icon2d()module.get\_icon2d()**

---

Retourne l'icône du module.

```
function get_icon2d( )
```

L'icone est au format PNG et a une taille maximale de 1536 octets.

**Retourne :**

un buffer binaire contenant l'icone, au format png.

**module→get\_lastLogs()**  
**module→lastLogs()module.get\_lastLogs()**

---

**YModule**

Retourne une chaîne de caractère contenant les derniers logs du module.

**function 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.

---

<b>module→get_logicalName()</b>	<b>YModule</b>
<b>module→logicalName()module.get_logicalName()</b>	

---

Retourne le nom logique du module.

```
function 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).

**function get\_luminosity( )**

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne Y\_LUMINOSITY\_INVALID.

**module→get\_persistentSettings()**  
**module→persistentSettings()**  
**module.get\_persistentSettings()**

**YModule**

Retourne l'état courant des réglages persistents du module.

```
function 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()**  
**module→productId()module.get\_productId()**

---

**YModule**

Retourne l'identifiant USB du module, préprogrammé en usine.

**function 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.

---

<b>module→get_productName()</b>	<b>YModule</b>
<b>module→productName()module.get_productName()</b>	

---

Retourne le nom commercial du module, préprogrammé en usine.

```
function 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.

**function 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()**  
**module→rebootCountdown()**  
**module.get\_rebootCountdown()**

**YModule**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

function **get\_rebootCountdown( )**

**Retourne :**

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

En cas d'erreur, déclenche une exception ou retourne **Y\_REBOOTCOUNTDOWN\_INVALID**.

**module→get\_serialNumber()** **YModule**  
**module→serialNumber()module.get\_serialNumber()**

---

Retourne le numéro de série du module, préprogrammé en usine.

```
function 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.

---

<b>module-&gt;get_upTime()</b>	<b>YModule</b>
<b>module-&gt;upTime()module.get_upTime()</b>	

---

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

```
function get_upTime( )
```

**Retourne :**

un entier représentant le nombre de millisecondes écoulées depuis la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_UPTIME\_INVALID.

**module→get\_usbBandwidth()**  
**module→usbBandwidth()**  
**module.get\_usbBandwidth()**

**YModule**

Retourne le nombre d'interface USB utilisé par le module.

```
function 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.

---

<b>module→get_usbCurrent()</b>	<b>YModule</b>
<b>module→usbCurrent()</b> <b>module.get_usbCurrent()</b>	

---

Retourne le courant consommé par le module sur le bus USB, en milliampères.

```
function 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).

```
function 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()|module.isOnline()****YModule**

Vérifie si le module est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()module.isOnline\_async()****YModule**

Vérifie si le module est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**module→load()module.load()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→load\_async()module.load\_async()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**module→nextModule()|module.nextModule()****YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

```
function nextModule( )
```

**Retourne :**

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**module→reboot()module.reboot()****YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
function reboot( secBeforeReboot)
```

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→revertFromFlash()****YModule****module.revertFromFlash()**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

```
function revertFromFlash( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## module→saveToFlash()module.saveToFlash()

YModule

Sauve les réglages courants dans la mémoire non volatile du module.

```
function 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()module.set_beacon()</b>	

---

Allume ou éteint la balise de localisation du module.

```
function set_beacon( newval)
```

**Paramètres :**

**newval** soit Y\_BEACON\_OFF, soit Y\_BEACON\_ON

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module->set\_logicalName()** YModule  
**module->setLogicalName()module.set\_logicalName()**

---

Change le nom logique du module.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>module-&gt;set_luminosity()</b>	<b>YModule</b>
<b>module-&gt;setLuminosity()module.set_luminosity()</b>	

---

Modifie la luminosité des leds informatives du module.

function **set\_luminosity( newval )**

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la luminosité des leds informatives du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module->set\_usbBandwidth()**  
**module->setUsbBandwidth()**  
**module.set\_usbBandwidth()**

**YModule**

Modifie le nombre d'interface USB utilisé par le module.

function **set\_usbBandwidth( newval )**

Vous devez redémarrer le module après avoir changé ce réglage.

**Paramètres :**

**newval** soit Y\_USBBANDWIDTH\_SIMPLE, soit Y\_USBBANDWIDTH\_DOUBLE, selon le nombre d'interface USB utilisé par le module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→set(userData)****YModule****module→setUserData()module.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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.

**function triggerFirmwareUpdate( secBeforeReboot )**

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→wait\_async()module.wait\_async()****YModule**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.26. Interface de la fonction Network

Les objets YNetwork permettent de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_network.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YNetwork = yoctolib.YNetwork;
php	require_once('yocto_network.php');
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()

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

#### **network→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_friendlyName()**

Retourne un identifiant global de l'interface réseau au format NOM\_MODULE . NOM\_FONCTION.

#### **network→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **network→get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

#### **network→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL . FUNCTIONID.

#### **network→get\_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

#### **network→get\_logicalName()**

Retourne le nom logique de l'interface réseau.

#### **network→get\_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

#### **network→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

#### **network→get\_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

#### **network→get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

#### **network→get\_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

#### **network→get\_secondaryDNS()**

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

#### **network→get\_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

#### **network→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **network→get\_userPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

#### **network→get\_wwwWatchdogDelay()**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

#### **network→isOnline()**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

### 3. Reference

<b>network→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.
<b>network→load(msValidity)</b>
Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.
<b>network→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.
<b>network→nextNetwork()</b>
Continue l'énumération des interfaces réseau commencée à l'aide de <code>yFirstNetwork()</code> .
<b>network→ping(host)</b>
Ping <code>str_host</code> pour vérifier la connexion réseau.
<b>network→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>network→set_adminPassword(newval)</b>
Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.
<b>network→set_callbackCredentials(newval)</b>
Modifie le laisser-passer pour se connecter à l'adresse de callback.
<b>network→set_callbackEncoding(newval)</b>
Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.
<b>network→set_callbackMaxDelay(newval)</b>
Modifie l'attente maximale entre deux notifications par callback, en secondes.
<b>network→set_callbackMethod(newval)</b>
Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.
<b>network→set_callbackMinDelay(newval)</b>
Modifie l'attente minimale entre deux notifications par callback, en secondes.
<b>network→set_callbackUrl(newval)</b>
Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.
<b>network→set_discoverable(newval)</b>
Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).
<b>network→set_logicalName(newval)</b>
Modifie le nom logique de l'interface réseau.
<b>network→set_primaryDNS(newval)</b>
Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.
<b>network→set_secondaryDNS(newval)</b>
Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.
<b>network→set_userData(data)</b>
Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get(userData)</code> .
<b>network→set_userPassword(newval)</b>
Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.
<b>network→set_wwwWatchdogDelay(newval)</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.
<b>network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)</b>

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() yFindNetwork()yFindNetwork()

**YNetwork**

Permet de retrouver une interface réseau d'après un identifiant donné.

```
function yFindNetwork( 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.

```
function 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.

```
function callbackLogin( username, password)
```

Le mot de passe ne sera pas stocké dans le module, mais seulement une version hashée non réversible. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**username** nom d'utilisateur pour s'identifier au callback

**password** mot de passe pour s'identifier au callback

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→describe()network.describe()****YNetwork**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**function 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 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.

**function 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).

```
function 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()**  
**network→callbackCredentials()**  
**network.get\_callbackCredentials()****YNetwork**

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

**function 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.

```
function 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.

```
function 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.

```
function get_callbackMethod( )
```

**Retourne :**

une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMETHOD\_INVALID.

**network→get\_callbackMinDelay()**  
**network→callbackMinDelay()**  
**network.get\_callbackMinDelay()**

---

**YNetwork**

Retourne l'attente minimale entre deux notifications par callback, en secondes.

```
function get_callbackMinDelay( )
```

**Retourne :**

un entier représentant l'attente minimale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMINDELAY\_INVALID.

**network→get\_callbackUrl()****YNetwork****network→callbackUrl()network.get\_callbackUrl()**

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
function get_callbackUrl( )
```

**Retourne :**

une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKURL\_INVALID.

**network→get\_discoverable()**

**YNetwork**

**network→discoverable()network.get\_discoverable()**

---

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

**function 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 (protocols 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.

```
function 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.

```
function 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.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**network→get\_functionId()****YNetwork****network→functionId()network.get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**network→get\_hardwareId()**

**YNetwork**

**network→hardwareId()network.get\_hardwareId()**

---

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL.FUNCTIONID.

**function 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.

```
function get_ipAddress( )
```

Il peut s'agir d'une adresse configurée statiquement, ou d'une adresse reçue par un serveur DHCP.

**Retourne :**

une chaîne de caractères représentant l'adresse IP utilisée par le module Yoctopuce

En cas d'erreur, déclenche une exception ou retourne Y\_IPADDRESS\_INVALID.

**network→get\_logicalName()**

**YNetwork**

**network→logicalName()network.get\_logicalName()**

---

Retourne le nom logique de l'interface réseau.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

**network→get\_macAddress()** **YNetwork**  
**network→macAddress()network.get\_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

```
function 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.

```
function 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\_module\_async()**  
**network→module\_async()**  
**network.get\_module\_async()**

**YNetwork**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module\_async( callback, context)**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**network→get\_poeCurrent()**

**YNetwork**

**network→poeCurrent()network.get\_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

```
function 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.

---

<b>network→get_primaryDNS()</b>	<b>YNetwork</b>
<b>network→primaryDNS()network.get_primaryDNS()</b>	

---

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
function 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.

```
function 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.

**network→get\_router()****YNetwork****network→router()network.get\_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

function **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.

```
function 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.

---

<b>network→get_subnetMask()</b>	<b>YNetwork</b>
<b>network→subnetMask()network.get_subnetMask()</b>	

---

Retourne le masque de sous-réseau utilisé par le module.

```
function 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).

```
function 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.

```
function get_userPassword( )
```

**Retourne :**

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_USERPASSWORD\_INVALID.

**network→get\_wwwWatchdogDelay()**  
**network→wwwWatchdogDelay()**  
**network.get\_wwwWatchdogDelay()**

**YNetwork**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

**function 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()network.isOnline()****YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()network.isOnline\_async()****YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**network→load()network.load()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→load\_async()|network.load\_async()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**network→nextNetwork()network.nextNetwork()****YNetwork**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

```
function 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.

```
function ping( 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**network→setAdminPassword()**  
**network.set\_adminPassword()**

**YNetwork**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

function **set\_adminPassword( newval )**

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackCredentials()**  
**network→setCallbackCredentials()**  
**network.set\_callbackCredentials()**

**YNetwork**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

**function set\_callbackCredentials( newval)**

Le laisser-passer doit être fourni tel que retourné par la fonction `get_callbackCredentials`, sous la forme `username:hash`. La valeur du hash dépend de la méthode d'autorisation implémentée par le callback. Pour une autorisation de type Basic, le hash est le MD5 de la chaîne `username:password`. Pour une autorisation de type Digest, le hash est le MD5 de la chaîne `username:realm:password`. Pour une utilisation simplifiée, utilisez la fonction `callbackLogin`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le laisser-passer pour se connecter à l'adresse de callback

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackEncoding()**  
**network→setCallbackEncoding()**  
**network.set\_callbackEncoding()**

**YNetwork**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
function set_callbackEncoding( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMaxDelay()**  
**network→setCallbackMaxDelay()**  
**network.set\_callbackMaxDelay()**

**YNetwork**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

```
function set_callbackMaxDelay( newval)
```

**Paramètres :**

**newval** un entier représentant l'attente maximale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMethod()**  
**network→setCallbackMethod()**  
**network.set\_callbackMethod()**

**YNetwork**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
function set_callbackMethod( newval)
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMinDelay()**  
**network→setCallbackMinDelay()**  
**network.set\_callbackMinDelay()**

**YNetwork**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

```
function set_callbackMinDelay( newval)
```

**Paramètres :**

**newval** un entier représentant l'attente minimale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## network→set\_callbackUrl()

YNetwork

## network→setCallbackUrl()network.set\_callbackUrl()

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
function set_callbackUrl( newval)
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>network→set_discoverable()</b>	<b>YNetwork</b>
<b>network→setDiscoverable()</b>	
<b>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).

function **set\_discoverable( newval )**

**Paramètres :**

**newval** soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (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.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**network→set\_primaryDNS()** YNetwork  
**network→setPrimaryDNS()network.set\_primaryDNS()**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
function set_primaryDNS( newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_secondaryDNS()**  
**network→setSecondaryDNS()**  
**network.set\_secondaryDNS()**

**YNetwork**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

**function set\_secondaryDNS( newval)**

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set(userData)****YNetwork****network→setUserData()network.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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.

function **set\_userPassword( newval)**

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_wwwWatchdogDelay()**  
**network→setWwwWatchdogDelay()**  
**network.set\_wwwWatchdogDelay()**

**YNetwork**

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

function **set\_wwwWatchdogDelay( newval )**

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW. La plus petite durée non-nulle utilisable est 90 secondes.

**Paramètres :**

**newval** un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useDHCP()network.useDHCP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

```
function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter )
```

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**fallbackIpAddr**              adresse IP à utiliser si aucun serveur DHCP ne répond  
**fallbackSubnetMaskLen**      longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.  
**fallbackRouter**              adresse de la passerelle à utiliser si aucun serveur DHCP ne répond

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useStaticIP()network.useStaticIP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

```
function useStaticIP( ipAddress, subnetMaskLen, router)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ipAddress**      adresse IP à utiliser par le module

**subnetMaskLen** longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.

**router**          adresse IP de la passerelle à utiliser ("default gateway")

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→wait\_async()|network.wait\_async()****YNetwork**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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;
php require_once('yocto_oscontrol.php');
cpp #include "yocto_oscontrol.h"
m #import "yocto_oscontrol.h"
pas uses yocto_oscontrol;
vb yocto_oscontrol.vb
cs yocto_oscontrol.cs
java import com.yoctopuce.YoctoAPI.YOsControl;
py from yocto_oscontrol import *

```

### 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)

### 3. Reference

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 contrôles 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.

function **yFindOsControl( 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() yFirstOsControl()yFirstOsControl()**

---

**YOsControl**

Commence l'énumération des contrôle d'OS accessibles par la librairie.

```
function 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()oscontrol.describe()****YOscControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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'OS (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**oscontrol→get\_advertisedValue()**  
**oscontrol→advertisedValue()**  
**oscontrol.get\_advertisedValue()**

**YOsControl**

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

**function get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**oscontrol→get\_errorMessage()**  
**oscontrol→errorMessage()**  
**oscontrol.get\_errorMessage()****YOscControl**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

**oscontrol→get\_errorType()**

**YOsControl**

**oscontrol→errorType()oscontrol.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

**oscontrol→get\_friendlyName()**  
**oscontrol→friendlyName()**  
**oscontrol.get\_friendlyName()****YOsControl**

Retourne un identifiant global du contrôle d'OS au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**oscontrol→get\_functionDescriptor()**  
**oscontrol→functionDescriptor()**  
**oscontrol.get\_functionDescriptor()**

---

**YOsControl**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function 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>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.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**oscontrol→get\_hardwareId()**

**YOsControl**

**oscontrol→hardwareId()oscontrol.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du contrôle d'OS au format SERIAL.FUNCTIONID.

**function 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.

```
function 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**oscontrol→get\_module\_async()**  
**oscontrol→module\_async()**  
**oscontrol.get\_module\_async()****YOsControl**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**oscontrol→get\_shutdownCountdown()**  
**oscontrol→shutdownCountdown()**  
**oscontrol.get\_shutdownCountdown()**

**YOsControl**

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

**function 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).

```
function 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()  
oscontrol.isOnline()****YOsControl**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()  
oscontrol.isOnline\_async()****YOsControl**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**oscontrol→load()oscontrol.load()****YOsControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**oscontrol→load\_async()oscontrol.load\_async()****YOsControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**oscontrol→nextOsControl()**  
**oscontrol.nextOsControl()**

---

**YOsControl**

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

**function nextOsControl( )**

**Retourne :**

un pointeur sur un objet `YOsControl` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**oscontrol→registerValueCallback()  
oscontrol.registerValueCallback()****YOscControl**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**oscontrol→set\_logicalName()**  
**oscontrol→setLogicalName()**  
**oscontrol.set\_logicalName()**

**YOsControl**

Modifie le nom logique du contrôle d'OS.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'OS.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**oscontrol→set(userData)****YOsControl****oscontrol→setUserData()oscontrol.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**oscontrol→shutdown()oscontrol.shutdown()****YOsControl**

Agende un arrêt de l'OS dans un nombre donné de secondes.

```
function shutdown( secBeforeShutDown)
```

**Paramètres :**

**secBeforeShutDown** nombre de secondes avant l'arrêt

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**oscontrol→wait\_async()****YOsControl**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.28. Interface de la fonction Power

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_power.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPower = yoctolib.YPower;
php require_once('yocto_power.php');
cpp #include "yocto_power.h"
m #import "yocto_power.h"
pas uses yocto_power;
vb yocto_power.vb
cs yocto_power.cs
java import com.yoctopuce.YoctoAPI.YPower;
py from yocto_power import *

```

### 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 renvoyée par le capteur (sans arrondi ni calibration).

#### **power→get\_currentValue()**

Retourne la valeur instantanée de la puissance électrique.

#### **power→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

#### **power→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

#### **power→get\_friendlyName()**

Retourne un identifiant global du capteur de puissance électrique au format NOM\_MODULE.NOM\_FONCTION.

#### **power→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**power→get\_functionId()**

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

**power→get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

**power→get\_highestValue()**

Retourne la valeur maximale observée pour la puissance électrique.

**power→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**power→get\_logicalName()**

Retourne le nom logique du capteur de puissance électrique.

**power→get\_lowestValue()**

Retourne la valeur minimale observée pour la puissance électrique.

**power→get\_meter()**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

**power→get\_meterTimer()**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

**power→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**power→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**power→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**power→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**power→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**power→get\_unit()**

Retourne l'unité dans laquelle la puissance électrique est exprimée.

**power→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**power→isOnline()**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

**power→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

**power→load(msValidity)**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

**power→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

**power→load\_async(msValidity, callback, context)**

### 3. Reference

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)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

#### **power→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPower.FindPower() yFindPower()yFindPower()

YPower

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

```
function yFindPower( 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() yFirstPower()yFirstPower()**

**YPower**

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

```
function 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.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→describe()power.describe()****YPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format TYPE (NAME )=SERIAL .FUNCTIONID.

**function 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 puissance électrique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**power→get\_advertisedValue()**  
**power→advertisedValue()**  
**power.get\_advertisedValue()**

**YPower**

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

```
function 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()**  
**power→cosPhi()power.get\_cosPhi()****YPower**

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

```
function 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()**  
**power→currentRawValue()**  
**power.get\_currentRawValue()****YPower**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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()**

**YPower**

**power→currentValue()power.get\_currentValue()**

---

Retourne la valeur instantanée de la puissance électrique.

```
function 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()****YPower****power→errorMessage()power.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
function 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.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

**power→get\_friendlyName()****YPower****power→friendlyName()power.get\_friendlyName()**

Retourne un identifiant global du capteur de puissance électrique au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**power→get\_functionId()****YPower****power→functionId()power.get\_functionId()**

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

```
function 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()**  
**power→hardwareId()power.get\_hardwareId()****YPower**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

**function 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.

```
function get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**power→get\_logFrequency()** YPower  
**power→logFrequency()** `power.get_logFrequency()`

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function 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>power→get_logicalName()</b>	<b>YPower</b>
<b>power→logicalName()power.get_logicalName()</b>	

---

Retourne le nom logique du capteur de puissance électrique.

```
function 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.

```
function 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.

---

<b>power→get_meter()</b>	<b>YPower</b>
<b>power→meter()power.get_meter()</b>	

---

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

```
function 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

```
function get_meterTimer( )
```

**Retourne :**

un entier représentant le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_METERTIMER\_INVALID.

**power→get\_module()****YPower****power→module()power.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function 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\_module\_async()** YPower  
**power→module\_async()power.get\_module\_async()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

<b>power→get_recordedData()</b>	<b>YPower</b>
<b>power→recordedData()power.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.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

#### Paramètres :

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

#### Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**power→get\_reportFrequency()**  
**power→reportFrequency()**  
**power.get\_reportFrequency()**

**YPower**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **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()**  
**power→resolution()power.get\_resolution()****YPower**

Retourne la résolution des valeurs mesurées.

```
function 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()**  
**power→unit()power.get\_unit()**

---

YPower

Retourne l'unité dans laquelle la puissance électrique est exprimée.

```
function 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).

```
function 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()power.isOnline()****YPower**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()|power.isOnline\_async()****YPower**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen  
**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**power→load()power.load()****YPower**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→loadCalibrationPoints()**  
**power.loadCalibrationPoints()****YPower**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→load\_async()power.load\_async()**

YPower

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**power→nextPower()power.nextPower()****YPower**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function reset( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_highestValue()****YPower****power→setHighestValue()power.set\_highestValue()**

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la puissance électrique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_logFrequency()** YPower  
**power→setLogFrequency()power.set\_logFrequency()**

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**power→set\_logicalName()** YPower  
**power→setLogicalName()power.set\_logicalName()**

Modifie le nom logique du capteur de puissance électrique.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de puissance 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.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la puissance électrique

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_reportFrequency()  
power→setReportFrequency()  
power.set\_reportFrequency()****YPower**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set\_resolution()** YPower  
**power→setResolution()power.set\_resolution()**

---

Modifie la résolution des valeurs mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→set(userData)****YPower****power→setUserData()power.set(userData())**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**power→wait\_async()power.wait\_async()**

YPower

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.29. Interface de la fonction Pressure

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_pressure.js'></script>
node.js	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 retournée par le capteur (sans arrondi ni calibration).

#### pressure→get\_currentValue()

Retourne la mesure actuelle de la pression.

#### pressure→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### pressure→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### pressure→get\_friendlyName()

Retourne un identifiant global du capteur de pression au format NOM\_MODULE . NOM\_FONCTION.

#### pressure→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### pressure→get\_functionId()

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

#### pressure→get\_hardwareId()

### 3. Reference

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>

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**pressure→set\_logicalName(newval)**

Modifie le nom logique du capteur de pression.

**pressure→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la pression.

**pressure→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**pressure→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**pressure→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**pressure→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPressure.FindPressure() yFindPressure()yFindPressure()

YPressure

Permet de retrouver un capteur de pression d'après un identifiant donné.

**function yFindPressure( func)**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode YPressure.isOnline() pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de pression sans ambiguïté

### Retourne :

un objet de classe YPressure qui permet ensuite de contrôler le capteur de pression.

**YPressure.FirstPressure()****yFirstPressure()yFirstPressure()****YPressure**

Commence l'énumération des capteurs de pression accessibles par la librairie.

```
function 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()****YPressure****pressure.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**function calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→describe()pressure.describe()****YPressure**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 pression (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pressure→get\_advertisedValue()**  
**pressure→advertisedValue()**  
**pressure.get\_advertisedValue()**

**YPressure**

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

**function get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de pression (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**pressure→get\_currentRawValue()**  
**pressure→currentRawValue()**  
**pressure.get\_currentRawValue()**

**YPressure**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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()** YPressure  
**pressure→currentValue()pressure.get\_currentValue()**

---

Retourne la mesure actuelle de la pression.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la mesure actuelle de la pression

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**pressure→get\_errorMessage()**  
**pressure→errorMessage()**  
**pressure.get\_errorMessage()****YPressure**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

**pressure→get\_errorType()**

**YPressure**

**pressure→errorType()pressure.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

**pressure→get\_friendlyName()**  
**pressure→friendlyName()**  
**pressure.get\_friendlyName()**

**YPressure**

Retourne un identifiant global du capteur de pression au format NOM\_MODULE . NOM\_FONCTION.

**function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**pressure→get\_functionId()****YPressure****pressure→functionId()pressure.get\_functionId()**

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

```
function 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.

**function 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.

```
function 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.

**function 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.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de pression. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pressure→get\_lowestValue()**

**YPressure**

**pressure→lowestValue()pressure.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la pression.

```
function 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.

```
function 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\_module\_async()**  
**pressure→module\_async()**  
**pressure.get\_module\_async()**

YPressure

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

#### Paramètres :

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

#### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**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.

**function get\_recordedData( startTime, endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**pressure→get\_reportFrequency()**  
**pressure→reportFrequency()**  
**pressure.get\_reportFrequency()**

**YPressure**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne **Y\_REPORTFREQUENCY\_INVALID**.

**pressure→get\_resolution()**

**YPressure**

**pressure→resolution()pressure.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function 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.

**function 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).

```
function 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()pressure.isOnline()****YPressure**

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()  
pressure.isOnline\_async()****YPressure**

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pressure→load()**pressure.load()******YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→loadCalibrationPoints()****YPressure****pressure.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→load\_async()pressure.load\_async()****YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pressure→nextPressure()pressure.nextPressure()****YPressure**

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

function **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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la pression

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_logFrequency()**  
**pressure→setLogFrequency()**  
**pressure.set\_logFrequency()**

**YPressure**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_logicalName()**  
**pressure→setLogicalName()**  
**pressure.set\_logicalName()**

**YPressure**

Modifie le nom logique du capteur de pression.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de pression.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_lowestValue()**  
**pressure→setLowestValue()**  
**pressure.set\_lowestValue()**

YPressure

Modifie la mémoire de valeur minimale observée pour la pression.

```
function set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la pression

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>pressure→set_reportFrequency()</b>	<b>YPressure</b>
<b>pressure→setReportFrequency()</b>	
<b>pressure.set_reportFrequency()</b>	

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_resolution()****YPressure****pressure→setResolution()pressure.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

function **set\_resolution( newval )**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set(userData)**

**YPressure**

**pressure→setUserData()pressure.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**pressure→wait\_async()pressure.wait\_async()****YPressure**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.30. Interface de la fonction Pwm

La librairie de programmation Yoctopuce permet simplement de configurer, démarrer et arrêter le PWM.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_pwmoutput.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPwmOutput = yoctolib.YPwmOutput;
php require_once('yocto_pwmoutput.php');
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>	Démarre 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.

### 3. Reference

---

**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.

function **yFindPwmOutput( 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.

```
function 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()pwmoutput.describe()****YPwmOutput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 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.

```
function dutyCycleMove( target, ms_duration)
```

**Paramètres :**

**target** nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)

**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→get\_advertisedValue()**  
**pwmoutput→advertisedValue()**  
**pwmoutput.get\_advertisedValue()**

**YPwmOutput**

Retourne la valeur courante du PWM (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du PWM (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**pwmoutput→get\_dutyCycle()**

**YPwmOutput**

**pwmoutput→dutyCycle()pwmoutput.get\_dutyCycle()**

---

Retourne le duty cycle du PWM, en pour cents.

```
function get_dutyCycle( )
```

**Retourne :**

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne `Y_DUTYCYCLE_INVALID`.

**pwmoutput→get\_dutyCycleAtPowerOn()**  
**pwmoutput→dutyCycleAtPowerOn()**  
**pwmoutput.get\_dutyCycleAtPowerOn()**

**YPwmOutput**

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

function **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\_DUTYCYLEATPOWERON\_INVALID**.

**pwmoutput→get\_enabled()**

**YPwmOutput**

**pwmoutput→enabled()pwmoutput.get\_enabled()**

---

Retourne l'état de fonctionnement du PWM.

```
function 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.

---

<b>pwmoutput→get_enabledAtPowerOn()</b>	<b>YPwmOutput</b>
<b>pwmoutput→enabledAtPowerOn()</b>	
<b>pwmoutput.get_enabledAtPowerOn()</b>	

---

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

```
function 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→getErrorMessage()**  
**pwmoutput→errorMessage()**  
**pwmoutput.getErrorMessage()**

**YPwmOutput**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

**function 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 PWM.

**pwmoutput→get\_errorType()****YPwmOutput****pwmoutput→errorType()pwmoutput.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

**pwmoutput→get\_frequency()**

**YPwmOutput**

**pwmoutput→frequency()pwmoutput.get\_frequency()**

---

Retourne la fréquence du PWM en Hz.

**function 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()**  
**pwmoutput→friendlyName()**  
**pwmoutput.get\_friendlyName()**

**YPwmOutput**

Retourne un identifiant global du PWM au format NOM\_MODULE.NOM\_FONCTION.

**function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

`pwmoutput->get_functionId()`

`YPwmOutput`

`pwmoutput->functionId()pwmoutput.get_functionId()`

Retourne l'identifiant matériel du PWM, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pwmoutput→get\_hardwareId()**  
**pwmoutput→hardwareId()**  
**pwmoutput.get\_hardwareId()**

**YPwmOutput**

Retourne l'identifiant matériel unique du PWM au format SERIAL.FUNCTIONID.

```
function 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.

```
function 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**pwmoutput→get\_module\_async()**  
**pwmoutput→module\_async()**  
**pwmoutput.get\_module\_async()**

**YPwmOutput**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module\_async( callback, context)**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de **YModule**

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pwmoutput→get\_period()**

**YPwmOutput**

**pwmoutput→period()pwmoutput.get\_period()**

---

Retourne la période du PWM en millisecondes.

```
function get_period( )
```

**Retourne :**

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PERIOD\_INVALID.

**pwmoutput→get\_pulseDuration()**  
**pwmoutput→pulseDuration()**  
**pwmoutput.get\_pulseDuration()**

**YPwmOutput**

Retourne la longueur d'une impulsion du PWM en millisecondes.

```
function get_pulseDuration( )
```

**Retourne :**

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PULSEDURATION\_INVALID.

**pwmoutput→get(userData)**

**YPwmOutput**

**pwmoutput→userData()pwmoutput.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pwmoutput→isOnline()pwmoutput.isOnline()****YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()**  
**pwmoutput.isOnline\_async()****YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pwmoutput→load()pwmoutput.load()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

function **load( msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→load\_async()pwmoutput.load\_async()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pwmoutput→nextPwmOutput()**  
**pwmoutput.nextPwmOutput()****YPwmOutput**

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput( )`.

```
function 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.

```
function pulseDurationMove( ms_target, ms_duration)
```

N'importe quel changement de fréquence, duty cycle, période ou encore de longueur d'impulsion annulera tout processus de transition en cours.

**Paramètres :**

**ms\_target** nouvelle longueur des impulsions à la fin de la transition (nombre flottant, représentant la longueur en millisecondes)

**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## pwmoutput→registerValueCallback() pwmoutput.registerValueCallback()

YPwmOutput

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_dutyCycle( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM, en pour cents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_dutyCycleAtPowerOn()**  
**pwmoutput→setDutyCycleAtPowerOn()**  
**pwmoutput.set\_dutyCycleAtPowerOn()**

**YPwmOutput**

Modifie le duty cycle du PWM au démarrage du module.

**function set\_dutyCycleAtPowerOn( newval)**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM au démarrage du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_enabled()**

**YPwmOutput**

**pwmoutput→setEnabled()pwmoutput.set\_enabled()**

---

Démarre ou arrête le PWM.

```
function set_enabled( newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_enabledAtPowerOn()**  
**pwmoutput→setEnabledAtPowerOn()**  
**pwmoutput.set\_enabledAtPowerOn()**

**YPwmOutput**

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

```
function set_enabledAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du fonctionnement du PWM à la mise sous tension du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_frequency()**  
**pwmoutput→setFrequency()**  
**pwmoutput.set\_frequency()**

---

YPwmOutput

Modifie la fréquence du PWM.

```
function set_frequency( newval )
```

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

**Paramètres :**

**newval** un entier représentant la fréquence du PWM

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_logicalName()**  
**pwmoutput→setLogicalName()**  
**pwmoutput.set\_logicalName()****YPwmOutput**

Modifie le nom logique du PWM.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du PWM.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_period()**

**YPwmOutput**

**pwmoutput→setPeriod()pwmoutput.set\_period()**

---

Modifie la période du PWM.

```
function set_period( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la période du PWM

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_pulseDuration()**  
**pwmoutput→setPulseDuration()**  
**pwmoutput.set\_pulseDuration()**

YPwmOutput

Modifie la longueur des impulsion du PWM, en millisecondes.

**function set\_pulseDuration( newval)**

Attention la longueur d'un impulsion ne peut pas être plus grande que la période, dans la cas contraire, la longueur sera automatiquement tronqué à la période.

**Paramètres :**

**newval** une valeur numérique représentant la longueur des impulsion du PWM, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set(userData)**  
**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).

```
function set(userData) {
```

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

**pwmoutput→wait\_async()****YPwmOutput**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.31. Interface de la fonction PwmPowerSource

La librairie de programmation Yoctopuce permet de configurer la source de tension utilisée par tous les PWM situés sur un même module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_pwmpowersource.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPwmPowerSource = yoctolib.YPwmPowerSource;
php require_once('yocto_pwmpowersource.php');
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)**

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() yFindPwmPowerSource()yFindPwmPowerSource()

YPwmPowerSource

Permet de retrouver une source de tension d'après un identifiant donné.

function **yFindPwmPowerSource( 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()yFirstPwmPowerSource()****YPwmPowerSource**

Commence l'énumération des Source de tension accessibles par la librairie.

```
function 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.

**function 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 la source de tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pwmpowersource→get\_advertisedValue()**  
**pwmpowersource→advertisedValue()**  
**pwmpowersource.get\_advertisedValue()**

**YPwmPowerSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
function 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()**  
**pwmpowersource→errorMessage()**  
**pwmpowersource.get\_errorMessage()**

**YPwmPowerSource**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

**function 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.

**function 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()**  
**pwmpowersource→friendlyName()**  
**pwmpowersource.get\_friendlyName()**

**YPwmPowerSource**

Retourne un identifiant global de la source de tension au format NOM\_MODULE.NOM\_FONCTION.

**function 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.

**function 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

**pwmpowersource→get\_functionId()**  
**pwmpowersource→functionId()**  
**pwmpowersource.get\_functionId()**

**YPwmPowerSource**

---

Retourne l'identifiant matériel de la source de tension, sans référence au module.

**function get\_functionId( )**

Par example `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.

**function 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.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de la source de tension. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pwmpowersource→get\_module()**  
**pwmpowersource→module()**  
**pwmpowersource.get\_module()**

**YPwmPowerSource**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function 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`

**pwmpowersource→get\_module\_async()**  
**pwmpowersource→module\_async()**  
**pwmpowersource.get\_module\_async()**

**YPwmPowerSource**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pwmpowersource→get\_powerMode()**  
**pwmpowersource→powerMode()**  
**pwmpowersource.get\_powerMode()**

**YPwmPowerSource**

Retourne la source de tension utilisé par tous les PWM du même module.

```
function 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)**  
**pwmpowersource→userData()**  
**pwmpowersource.get(userData)**

**YPwmPowerSource**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function 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.

function **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→isOnline\_async()**  
**pwmpowersource.isOnline\_async()****YPwmPowerSource**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pwmpowersource→load()pwmpowersource.load()****YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

function **load( msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→load\_async()**  
**pwmpowersource.load\_async()****YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**pwmpowersource→nextPwmPowerSource()**  
**pwmpowersource.nextPwmPowerSource()****YPwmPowerSource**

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique de la source de tension.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→set\_powerMode()**  
**pwmpowersource→setPowerMode()**  
**pwmpowersource.set\_powerMode()**

**YPwmPowerSource**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

function **set\_powerMode( newval )**

Le PWM peut aussi en mode open drain, dans ce code il tire activement la ligne à zéro volts. Attention ce paramètre est commun à tous les PWM du module, si vous changez le valeur de ce paramètre, tous les PWM situés sur le même module seront affectés. Si vous souhaitez que le changement de ce paramètre soit conservé après un redémarrage du module, n'oubliez pas d'appeler la méthode `saveToFlash()`.

**Paramètres :**

**newval** une valeur parmi `Y_POWERMODE_USB_5V`, `Y_POWERMODE_USB_3V`, `Y_POWERMODE_EXT_V` et `Y_POWERMODE_OPNDRN` représentant le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→set(userData)**  
**pwmpowersource→setUserData()**  
**pwmpowersource.set(userData)**

**YPwmPowerSource**

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).

```
function set(userData) {
```

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

**pwmpowersource→wait\_async()  
pwmpowersource.wait\_async()****YPwmPowerSource**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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>
node.js 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

#### 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 renvoyée 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.

### 3. Reference

<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() yFindQt()yFindQt()

YQt

Permet de retrouver un élément de quaternion d'après un identifiant donné.

**function yFindQt( func )**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnline()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

### Retourne :

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

**YQt.FirstQt()****YQt****yFirstQt()yFirstQt()**

Commence l'énumération des éléments de quaternion accessibles par la librairie.

function **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.

function **calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**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.

function **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 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).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**qt→get\_currentRawValue()****YQt****qt→currentRawValue()qt.get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**qt→get\_currentValue()**  
**qt→currentValue()qt.get\_currentValue()**

---

YQt

Retourne la valeur actuelle de la coordonnée.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la coordonnée

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**qt→get\_errorMessage()****YQt****qt→errorMessage()qt.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

**qt→get\_errorType()****YQt****qt→errorType()qt.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

**qt→get\_friendlyName()****YQt****qt→friendlyName()qt.get\_friendlyName()**

Retourne un identifiant global de l'élément de quaternion au format NOM\_MODULE.NOM\_FONCTION.

```
function 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()** YQt  
**qt→functionDescriptor()qt.get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**qt→get\_functionId()****YQt****qt→functionId()qt.get\_functionId()**

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

```
function 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`.

---

<b>qt→get_hardwareId()</b>	<b>YQt</b>
<b>qt→hardwareId()qt.get_hardwareId()</b>	

---

Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL.FUNCTIONID.

```
function 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.

```
function 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()  
qt→logFrequency()qt.get\_logFrequency()****YQt**

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.

```
function 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.

```
function 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.

```
function 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.

```
function 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>qt→get_module_async()</b>	<b>YQt</b>
<b>qt→module_async()qt.get_module_async()</b>	

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**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.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**qt→get\_reportFrequency()****YQt****qt→reportFrequency()|qt.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**function get\_reportFrequency( )****Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**qt→get\_resolution()****YQt****qt→resolution()qt.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function 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()**  
**qt→unit()qt.get\_unit()**

---

**YQt**

Retourne l'unité dans laquelle la coordonnée est exprimée.

**function 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).

```
function 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()qt.isOnline()****YQt**

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()|qt.isOnline\_async()**

YQt

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**qt→load()qt.load()****YQt**

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

**function load( msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→loadCalibrationPoints()|qt.loadCalibrationPoints()****YQt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## qt→load\_async()qt.load\_async()

YQt

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**qt→nextQt()qt.nextQt()****YQt**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**qt→set\_highestValue()**  
**qt→setHighestValue()qt.set\_highestValue()**

---

YQt

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logFrequency()**

YQt

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>qt→set_logicalName()</b>	<b>YQt</b>
<b>qt→setLogicalName()qt.set_logicalName()</b>	

---

Modifie le nom logique de l'élément de quaternion.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'élément de quaternion.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**qt→set\_lowestValue()**  
**qt→setLowestValue()qt.set\_lowestValue()****YQt**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_reportFrequency()** **YQt**  
**qt→setReportFrequency()qt.set\_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_resolution()****YQt****qt→setResolution()qt.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

function **set\_resolution( newval )**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set(userData)** YQt

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).

```
function set(userData) {
```

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

**qt→wait\_async()qt.wait\_async()****YQt**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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_realtimeclock.js'></script>
nodejs var yoctolib = require('yoctolib');
var YRealTimeClock = yoctolib.YRealTimeClock;
php require_once('yocto_realtimeclock.php');
cpp #include "yocto_realtimeclock.h"
m #import "yocto_realtimeclock.h"
pas uses yocto_realtimeclock;
vb yocto_realtimeclock.vb
cs yocto_realtimeclock.cs
java import com.yoctopuce.YoctoAPI.YRealTimeClock;
py from yocto_realtimeclock import *

```

### Fonction globales

#### 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()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**realtimeclock→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**realtimeclock→get\_timeSet()**

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

**realtimeclock→get\_unixTime()**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

**realtimeclock→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**realtimeclock→get\_utcOffset()**

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**realtimeclock→isOnline()**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**realtimeclock→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**realtimeclock→load(msValidity)**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**realtimeclock→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**realtimeclock→nextRealTimeClock()**

Continue l'énumération des horloge commencée à l'aide de yFirstRealTimeClock( ).

**realtimeclock→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**realtimeclock→set\_logicalName(newval)**

Modifie le nom logique de l'horloge.

**realtimeclock→set\_unixTime(newval)**

Modifie l'heure courante.

**realtimeclock→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**realtimeclock→set\_utcOffset(newval)**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**realtimeclock→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YRealTimeClock.FindRealTimeClock() yFindRealTimeClock()yFindRealTimeClock()

YRealTimeClock

Permet de retrouver une horloge d'après un identifiant donné.

```
function yFindRealTimeClock( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnline()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

`func` une chaîne de caractères qui référence l'horloge sans ambiguïté

**Retourne :**

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

**YRealTimeClock.FirstRealTimeClock()****yFirstRealTimeClock()yFirstRealTimeClock()****YRealTimeClock**

Commence l'énumération des horloge accessibles par la librairie.

```
function 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.

```
function 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()**  
**realtimeclock→advertisedValue()**  
**realtimeclock.get\_advertisedValue()**

**YRealTimeClock**

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**realtimeclock→getDateTime()**  
**realtimeclock→dateTime()**  
**realtimeclock.getDateTime()**

---

**YRealTimeClock**

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

```
function 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→get\_errorMessage()**  
**realtimeclock→errorMessage()**  
**realtimeclock.get\_errorMessage()**

**YRealTimeClock**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

**function get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

**realtimeclock→get\_errorType()**  
**realtimeclock→errorType()**  
**realtimeclock.get\_errorType()**

**YRealTimeClock**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

```
function 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()**  
**realtimeclock→friendlyName()**  
**realtimeclock.get\_friendlyName()**

**YRealTimeClock**

Retourne un identifiant global de l'horloge au format NOM\_MODULE.NOM\_FONCTION.

**function 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.

**realtimeclock→get\_functionDescriptor()**  
**realtimeclock→functionDescriptor()**  
**realtimeclock.get\_functionDescriptor()**

**YRealTimeClock**

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**realtimeclock→get\_functionId()**  
**realtimeclock→functionId()**  
**realtimeclock.get\_functionId()**

**YRealTimeClock**

Retourne l'identifiant matériel de l'horloge, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**realtimeclock→get\_hardwareId()**  
**realtimeclock→hardwareId()**  
**realtimeclock.get\_hardwareId()**

**YRealTimeClock**

Retourne l'identifiant matériel unique de l'horloge au format SERIAL.FUNCTIONID.

```
function 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.

**realtimeclock→get\_logicalName()**  
**realtimeclock→logicalName()**  
**realtimeclock.get\_logicalName()**

**YRealTimeClock**

Retourne le nom logique de l'horloge.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'horloge. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**realtimeclock→get\_module()** **YRealTimeClock**  
**realtimeclock→module()realtimeclock.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**realtimeclock→get\_module\_async()**  
**realtimeclock→module\_async()**  
**realtimeclock.get\_module\_async()**

**YRealTimeClock**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module\_async( callback, context)**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**realtimeclock→get\_timeSet()**

**YRealTimeClock**

**realtimeclock→timeSet()realtimeclock.get\_timeSet()**

---

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

```
function get_timeSet( )
```

**Retourne :**

soit Y\_TIMESET\_FALSE, soit Y\_TIMESET\_TRUE, selon vrai si l'horloge à été mise à l'heure, sinon faux

En cas d'erreur, déclenche une exception ou retourne Y\_TIMESET\_INVALID.

**realtimeclock→get\_unixTime()**  
**realtimeclock→unixTime()**  
**realtimeclock.get\_unixTime()**

**YRealTimeClock**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

function **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)**  
**realtimeclock→userData()**  
**realtimeclock.get(userData)**

**YRealTimeClock**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**realtimeclock→get\_utcOffset()**  
**realtimeclock→utcOffset()**  
**realtimeclock.get\_utcOffset()**

**YRealTimeClock**

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

```
function 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.

```
function 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→isOnline\_async()  
realtimeclock.isOnline\_async()****YRealTimeClock**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**realtimeclock→load()realtimeclock.load()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→load\_async()  
realtimeclock.load\_async()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**realtimeclock→nextRealTimeClock()**  
**realtimeclock.nextRealTimeClock()**

---

**YRealTimeClock**

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

**function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**realtimeclock→set\_logicalName()**  
**realtimeclock→setLogicalName()**  
**realtimeclock.set\_logicalName()**

**YRealTimeClock**

Modifie le nom logique de l'horloge.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'horloge.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→set\_unixTime()**  
**realtimeclock→setUnixTime()**  
**realtimeclock.set\_unixTime()**

**YRealTimeClock**

Modifie l'heure courante.

**function set\_unixTime( newval)**

L'heure est passée au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970). Si l'heure UTC est connue, l'attribut utcOffset sera automatiquement ajusté en fonction de l'heure configurée.

**Paramètres :**

**newval** un entier représentant l'heure courante

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→set(userData)**  
**realtimeclock→setUserData()**  
**realtimeclock.set(userData)**

**YRealTimeClock**

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).

```
function set(userData) {
```

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()**  
**realtimeclock→setUtcOffset()**  
**realtimeclock.set\_utcOffset()**

**YRealTimeClock**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**function set\_utcOffset( 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.

**realtimeclock→wait\_async()  
realtimeclock.wait\_async()****YRealTimeClock**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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>
node.js	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()

### 3. Reference

Retourne le cap de référence utilisé par le compas.
<b>refframe→get_errorMessage()</b> Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.
<b>refframe→get_errorType()</b> Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.
<b>refframe→get_friendlyName()</b> Retourne un identifiant global du référentiel au format NOM_MODULE . NOM_FONCTION.
<b>refframe→get_functionDescriptor()</b> Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>refframe→get_functionId()</b> Retourne l'identifiant matériel du référentiel, sans référence au module.
<b>refframe→get_hardwareId()</b> Retourne l'identifiant matériel unique du référentiel au format SERIAL . FUNCTIONID.
<b>refframe→get_logicalName()</b> Retourne le nom logique du référentiel.
<b>refframe→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>refframe→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<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.
<b>refframe→get_mountPosition()</b> Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.
<b>refframe→get(userData)</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>refframe→isOnline()</b> Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.
<b>refframe→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.
<b>refframe→load(msValidity)</b> Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.
<b>refframe→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.
<b>refframe→more3DCalibration()</b> Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.
<b>refframe→nextRefFrame()</b> Continue l'énumération des référentiels commencée à l'aide de yFirstRefFrame( ).
<b>refframe→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>refframe→save3DCalibration()</b> Applique les paramètres de calibration tridimensionnelle précédemment calculés.
<b>refframe→set_bearing(newval)</b>

Modifie le cap de référence utilisé par le compas.

**refframe→set\_logicalName(newval)**

Modifie le nom logique du référentiel.

**refframe→set\_mountPosition(position, orientation)**

Modifie le référentiel de la boussole et des inclinomètres.

**refframe→set(userData)**

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() yFindRefFrame()yFindRefFrame()

YRefFrame

Permet de retrouver un référentiel d'après un identifiant donné.

```
function yFindRefFrame( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode YRefFrame.isOnline() pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le référentiel sans ambiguïté

### Retourne :

un objet de classe YRefFrame qui permet ensuite de contrôler le référentiel.

**YRefFrame.FirstRefFrame()  
yFirstRefFrame()yFirstRefFrame()****YRefFrame**

Commence l'énumération des référentiels accessibles par la librairie.

```
function 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.

**function cancel3DCalibration( )**

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→describe()refframe.describe()****YRefFrame**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 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`.

```
function get_3DCalibrationHint( )
```

**Retourne :**

une chaîne de caractères.

**refframe→get\_3DCalibrationLogMsg()**  
**refframe→3DCalibrationLogMsg()**  
**refframe.get\_3DCalibrationLogMsg()**

**YRefFrame**

Retourne le dernier message de log produit par le processus de calibration.

**function 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.

**function 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.

function **get\_3DCalibrationStage( )**

**Retourne :**

une nombre entier, croissant au fur et à mesure de la compléion 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.

**function 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).

```
function 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.

**function 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→getErrorMessage()  
refframe→errorMessage()  
refframe.getErrorMessage()****YRefFrame**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

```
function 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 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.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

**refframe→get\_friendlyName()****YRefFrame****refframe→friendlyName()refframe.get\_friendlyName()**

Retourne un identifiant global du référentiel au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**refframe→get\_functionId()****YRefFrame****refframe→functionId()refframe.get\_functionId()**

Retourne l'identifiant matériel du référentiel, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**refframe→get\_hardwareId()**

**YRefFrame**

**refframe→hardwareId()refframe.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du référentiel au format SERIAL.FUNCTIONID.

**function 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()**

**YRefFrame**

**refframe→logicalName()refframe.get\_logicalName()**

---

Retourne le nom logique du référentiel.

```
function 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**refframe→get\_module\_async()**  
**refframe→module\_async()**  
**refframe.get\_module\_async()****YRefFrame**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

<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.

function **get\_mountOrientation( )**

**Retourne :**

une valeur parmi l'énumération `Y_MOUNTORIENTATION` (`Y_MOUNTORIENTATION_TWELVE`, `Y_MOUNTORIENTATION_THREE`, `Y_MOUNTORIENTATION_SIX`, `Y_MOUNTORIENTATION_NINE`) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get\_mountPosition()****YRefFrame****refframe→mountPosition()****refframe.get\_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

```
function 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).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**refframe→isOnline()refframe.isOnline()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**refframe→load()refframe.load()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

function **load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→load\_async()refframe.load\_async()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**refframe→more3DCalibration()****YRefFrame****refframe.more3DCalibration()**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

```
function 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()`.

```
function nextRefFrame( )
```

**Retourne :**

un pointeur sur un objet `YRefFrame` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**refframe→registerValueCallback()  
refframe.registerValueCallback()****YRefFrame**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**function 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.

```
function set_bearing( newval)
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici. Par exemple, si vous indiquez comme cap de référence la valeur de la déclinaison magnétique terrestre, le compas donnera l'orientation par rapport au Nord géographique. De même, si le capteur n'est pas positionné dans une des directions standard à cause d'un angle de lacet supplémentaire, vous pouvez le configurer comme cap de référence afin que le compas donne la direction naturelle attendue.

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur numérique représentant le cap de référence utilisé par le compas

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_logicalName()**  
**refframe→setLogicalName()**  
**refframe.set\_logicalName()**

**YRefFrame**

Modifie le nom logique du référentiel.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du référentiel.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_mountPosition()**  
**refframe→setMountPosition()**  
**refframe.set\_mountPosition()**

**YRefFrame**

Modifie le référentiel de la boussole et des inclinomètres.

**function set\_mountPosition( position, orientation)**

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas où le module n'est pas utilisé horizontalement et à l'endroit, il faut indiquer son orientation de référence (parallèle à la surface terrestre) afin que les mesures soient faites relativement à cette position.

#### Paramètres :

**position** une valeur parmi l'énumération Y\_MOUNTPOSITION (Y\_MOUNTPOSITION\_BOTTOM, Y\_MOUNTPOSITION\_TOP, Y\_MOUNTPOSITION\_FRONT, Y\_MOUNTPOSITION\_RIGHT, Y\_MOUNTPOSITION\_REAR, Y\_MOUNTPOSITION\_LEFT), correspondant à l'installation dans une boîte, sur l'une des six faces.

**orientation** une valeur parmi l'énumération Y\_MOUNTORIENTATION (Y\_MOUNTORIENTATION\_TWELVE, Y\_MOUNTORIENTATION\_THREE, Y\_MOUNTORIENTATION\_SIX, Y\_MOUNTORIENTATION\_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière. N'oubliez pas d'appeler la méthode saveToFlash() du module si le réglage doit être préservé.

**refframe→set(userData)**

**YRefFrame**

**refframe→setUserData()refframe.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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.

```
function 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.

**refframe→wait\_async(refframe.wait\_async())****YRefFrame**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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>	

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() yFindRelay()yFindRelay()

YRelay

Permet de retrouver un relais d'après un identifiant donné.

```
function yFindRelay( 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.

```
function 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()relay.delayedPulse()**

YRelay

Pré-programme une impulsion

```
function delayedPulse( ms_delay, ms_duration)
```

**Paramètres :**

**ms\_delay**      délai d'attente avant l'impulsion, en millisecondes

**ms\_duration**      durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## relay→describe()relay.describe()

## YRelay

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function 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 relais (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**relay→get\_advertisedValue()**

**YRelay**

**relay→advertisedValue()relay.get\_advertisedValue()**

---

Retourne la valeur courante du relais (pas plus de 6 caractères).

```
function 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().

```
function 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→getErrorMessage()**

YRelay

**relay→errorMessage()relay.getErrorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
function 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 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.

```
function 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.

---

<b>relay→get_friendlyName()</b>	<b>YRelay</b>
<b>relay→friendlyName()relay.get_friendlyName()</b>	

---

Retourne un identifiant global du relais au format NOM\_MODULE.NOM\_FONCTION.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du relais si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du relais (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le relais en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**relay→get\_functionDescriptor()**  
**relay→functionDescriptor()**  
**relay.get\_functionDescriptor()**

**YRelay**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function 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

**relay->get\_functionId()**

**YRelay**

**relay->functionId()relay.get\_functionId()**

---

Retourne l'identifiant matériel du relais, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le relais (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**relay→get\_hardwareId()****YRelay****relay→hardwareId()relay.get\_hardwareId()**

Retourne l'identifiant matériel unique du relais au format SERIAL.FUNCTIONID.

```
function 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.

```
function 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.

```
function 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.

```
function 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.

```
function 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\_module\_async()** YRelay  
**relay→module\_async()relay.get\_module\_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**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.

```
function 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.

```
function 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).

```
function 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).

```
function 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).

```
function 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()relay.isOnline()****YRelay**

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()relay.isOnline\_async()

## YRelay

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context )
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**relay→load()relay.load()****YRelay**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
function load( msValidity )
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## relay→load\_async()relay.load\_async()

## YRelay

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## **relay→nextRelay()|relay.nextRelay()**

**YRelay**

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

```
function 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).

```
function pulse( ms_duration )
```

### Paramètres :

**ms\_duration** durée de l'impulsion, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→registerValueCallback()**  
**relay.registerValueCallback()****YRelay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()****YRelay****relay→setLogicalName()relay.set\_logicalName()**

Modifie le nom logique du relais.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du relais.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_maxTimeOnStateA()**  
**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.

function **set\_maxTimeOnStateA( newval )**

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_maxTimeOnStateB()**  
**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.

```
function set_maxTimeOnStateB( newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_output()**

**YRelay**

**relay→setOutput()relay.set\_output()**

---

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
function set_output( newval)
```

**Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay->set\_state()****YRelay****relay->setState()relay.set\_state()**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
function set_state( newval)
```

**Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>relay→set_stateAtPowerOn()</b>	<b>YRelay</b>
<b>relay→setStateAtPowerOn()</b>	
<b>relay.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).

```
function set_stateAtPowerOn( newval )
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set(userData)****YRelay****relay→setUserData()relay.set(userData())**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**relay→wait\_async()relay.wait\_async()****YRelay**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.36. Interface des fonctions de type senseur

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_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 rentrée 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()**

### 3. Reference

Retourne l'identifiant matériel unique du senseur au format SERIAL . FUNCTIONID.
<b>sensor-&gt;get_highestValue()</b> Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.
<b>sensor-&gt;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-&gt;get_logicalName()</b> Retourne le nom logique du senseur.
<b>sensor-&gt;get_lowestValue()</b> Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.
<b>sensor-&gt;get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>sensor-&gt;get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>sensor-&gt;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-&gt;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-&gt;get_resolution()</b> Retourne la résolution des valeurs mesurées.
<b>sensor-&gt;get_unit()</b> Retourne l'unité dans laquelle la mesure est exprimée.
<b>sensor-&gt;get(userData)</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>sensor-&gt;isOnline()</b> Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.
<b>sensor-&gt;isOnline_async(callback, context)</b> Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.
<b>sensor-&gt;load(msValidity)</b> Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.
<b>sensor-&gt;loadCalibrationPoints(rawValues, refValues)</b> Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>sensor-&gt;load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.
<b>sensor-&gt;nextSensor()</b> Continue l'énumération des senseurs commencée à l'aide de yFirstSensor( ).
<b>sensor-&gt;registerTimedReportCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>sensor-&gt;registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>sensor-&gt;set_highestValue(newval)</b> Modifie la mémoire de valeur maximale observée.
<b>sensor-&gt;set_logFrequency(newval)</b>

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() yFindSensor() yFindSensor()

YSensor

Permet de retrouver un senseur d'après un identifiant donné.

```
function yFindSensor( 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()****YSensor****yFirstSensor()yFirstSensor()**

Commence l'énumération des senseurs accessibles par la librairie.

```
function 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.

**sensor→calibrateFromPoints()  
sensor.calibrateFromPoints()****YSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**function calibrateFromPoints( rawValues, refValues)**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→describe()sensor.describe()****YSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 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).

**function 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).

```
function 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()**

**YSensor**

**sensor→currentValue()sensor.get\_currentValue()**

---

Retourne la valeur actuelle de la mesure.

function **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→getErrorMessage()****YSensor****sensor→errorMessage()sensor.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

```
function 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 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.

**function 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.

---

<b>sensor→get_friendlyName()</b>	<b>YSensor</b>
<b>sensor→friendlyName()sensor.get_friendlyName()</b>	

---

Retourne un identifiant global du senseur au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

**sensor→get\_functionId()****YSensor****sensor→functionId()sensor.get\_functionId()**

---

Retourne l'identifiant matériel du senseur, sans référence au module.

```
function 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`.

<b>sensor→get_hardwareId()</b>	<b>YSensor</b>
<b>sensor→hardwareId()sensor.get_hardwareId()</b>	

---

Retourne l'identifiant matériel unique du senseur au format SERIAL.FUNCTIONID.

```
function 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.

---

<b>sensor→get_highestValue()</b>	<b>YSensor</b>
<b>sensor→highestValue()sensor.get_highestValue()</b>	

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

```
function get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**sensor→get\_logFrequency()**

**YSensor**

**sensor→logFrequency()sensor.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function 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.

```
function get_logicalName( )
```

**Retourne :**

une chaîne de caractères représentant le nom logique du senseur. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**sensor→get\_lowestValue()**

**YSensor**

**sensor→lowestValue()sensor.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

**function 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.

```
function 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>sensor→get_module_async()</b>	<b>YSensor</b>
<b>sensor→module_async()sensor.get_module_async()</b>	

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

<b>sensor→get_recordedData()</b>	<b>YSensor</b>
<b>sensor→recordedData()sensor.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.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

#### Paramètres :

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

#### Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**sensor→get\_reportFrequency()**  
**sensor→reportFrequency()**  
**sensor.get\_reportFrequency()**

**YSensor**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne **Y\_REPORTFREQUENCY\_INVALID**.

---

**sensor→get\_resolution()****YSensor****sensor→resolution()sensor.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

```
function 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.

**function 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).

```
function 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()sensor.isOnline()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()sensor.isOnline\_async()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**sensor→load()sensor.load()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→loadCalibrationPoints()**  
**sensor.loadCalibrationPoints()****YSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→load\_async()sensor.load\_async()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**sensor→nextSensor()|sensor.nextSensor()****YSensor**

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

**sensor→registerValueCallback()**  
**sensor.registerValueCallback()****YSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_logFrequency()**  
**sensor→setLogFrequency()**  
**sensor.set\_logFrequency()**

**YSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_logicalName()** YSensor  
**sensor→setLogicalName()sensor.set\_logicalName()**

---

Modifie le nom logique du senseur.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du senseur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>sensor→set_lowestValue()</b>	<b>YSensor</b>
<b>sensor→setLowestValue()sensor.set_lowestValue()</b>	

---

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_reportFrequency()**  
**sensor→setReportFrequency()**  
**sensor.set\_reportFrequency()**

**YSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_resolution()****YSensor****sensor→setResolution()sensor.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

function **set\_resolution( newval )**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set(userData)**

**YSensor**

**sensor→setUserData()sensor.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**sensor→wait\_async()|sensor.wait\_async()****YSensor**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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>
nodejs 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.

**`servo→get_module()`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`servo→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`servo→get_neutral()`**

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

**`servo→get_position()`**

Retourne la position courante du servo.

**`servo→get_positionAtPowerOn()`**

Retourne la position du servo au démarrage du module.

**`servo→get_range()`**

Retourne la plage d'utilisation du servo.

**`servo→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set(userData)`.

**`servo→isOnline()`**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

**`servo→isOnline_async(callback, context)`**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

**`servo→load(msValidity)`**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

**`servo→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

**`servo→move(target, ms_duration)`**

Déclenche un mouvement à vitesse constante vers une position donnée.

**`servo→nextServo()`**

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

**`servo→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`servo→set_enabled(newval)`**

Démarre ou arrête le \$FUNCTION\$.

**`servo→set_enabledAtPowerOn(newval)`**

Configure l'état du générateur de signal de commande du servo au démarrage du module.

**`servo→set_logicalName(newval)`**

Modifie le nom logique du servo.

**`servo→set_neutral(newval)`**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

**`servo→set_position(newval)`**

Modifie immédiatement la consigne de position du servo.

**`servo→set_positionAtPowerOn(newval)`**

Configure la position du servo au démarrage du module.

**`servo→set_range(newval)`**

Modifie la plage d'utilisation du servo, en pourcents.

**`servo→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**`servo→wait_async(callback, context)`**

### **3. Reference**

---

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YServo.FindServo() yFindServo()yFindServo()

YServo

Permet de retrouver un servo d'après un identifiant donné.

function **yFindServo( func )**

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le servo soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le servo sans ambiguïté

**Retourne :**

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

## YServo.FirstServo() yFirstServo()yFirstServo()

YServo

Commence l'énumération des servo accessibles par la librairie.

```
function 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()servo.describe()****YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format TYPE (NAME )=SERIAL.FUNCTIONID.

function **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 servo (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**servo→get\_advertisedValue()**

**YServo**

**servo→advertisedValue()servo.get\_advertisedValue()**

---

Retourne la valeur courante du servo (pas plus de 6 caractères).

```
function 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\$.

```
function get_enabled( )
```

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du \$FUNCTION\$

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**servo→get\_enabledAtPowerOn()**  
**servo→enabledAtPowerOn()**  
**servo.get\_enabledAtPowerOn()**

**YServo**

---

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

```
function 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()****YServo****servo→errorMessage()servo.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
function 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()**  
**servo→errorType()servo.get\_errorType()**

**YServo**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du servo.

**servo→get\_friendlyName()****YServo****servo→friendlyName()servo.get\_friendlyName()**

Retourne un identifiant global du servo au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**servo→get\_functionId()****YServo****servo→functionId()servo.get\_functionId()**

Retourne l'identifiant matériel du servo, sans référence au module.

```
function 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.

**function 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.

```
function 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.

```
function 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>servo→get_module_async()</b>	<b>YServo</b>
<b>servo→module_async()servo.get_module_async()</b>	

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**servo→get\_neutral()**

**YServo**

**servo→neutral()servo.get\_neutral()**

---

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

function **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.

```
function 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.

```
function 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.

```
function 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).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**servo→isOnline()servo.isOnline()****YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()servo.isOnline\_async()****YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**servo→load()servo.load()****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→load\_async()servo.load\_async()****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**servo→move()servo.move()****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

```
function move( target, ms_duration)
```

**Paramètres :**

**target** nouvelle position à la fin du mouvement

**ms\_duration** durée totale du mouvement, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **servo→nextServo()servo.nextServo()**

**YServo**

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

```
function nextServo( )
```

**Retourne :**

un pointeur sur un objet `YServo` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**servo→registerValueCallback()  
servo.registerValueCallback()****YServo**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**servo→setEnabled()servo.set\_enabled()**

---

YServo

Démarre ou arrête le \$FUNCTION\$.

```
function set_enabled( newval)
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_enabledAtPowerOn()**  
**servo→setEnabledAtPowerOn()**  
**servo.set\_enabledAtPowerOn()**

YServo

Configure l'état du générateur de signal de commande du servo au démarrage du module.

function **set\_enabledAtPowerOn( newval )**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>servo→set_logicalName()</b>	<b>YServo</b>
<b>servo→setLogicalName()servo.set_logicalName()</b>	

---

Modifie le nom logique du servo.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du servo.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_neutral()****YServo****servo→setNeutral()servo.set\_neutral()**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
function set_neutral( newval)
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_position()  
servo→setPosition()servo.set\_position()**

---

YServo

Modifie immédiatement la consigne de position du servo.

```
function set_position( newval )
```

**Paramètres :**

**newval** un entier représentant immédiatement la consigne de position du servo

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_positionAtPowerOn()**  
**servo→setPositionAtPowerOn()**  
**servo.set\_positionAtPowerOn()**

YServo

Configure la position du servo au démarrage du module.

function **set\_positionAtPowerOn( newval )**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** un entier

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_range()  
servo→setRange()servo.set\_range()****YServo**

Modifie la plage d'utilisation du servo, en pourcents.

```
function set_range( newval)
```

La valeur 100% correspond à un signal de commande standard, variant de 1 [ms] à 2 [ms]. Pour les servos supportent une plage double, de 0.5 [ms] à 2.5 [ms], vous pouvez utiliser une valeur allant jusqu'à 200%. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la plage d'utilisation du servo, en pourcents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set(userData)****YServo****servo→setUserData()servo.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**servo→wait\_async()servo.wait\_async()****YServo**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.38. Interface de la fonction Temperature

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_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 retournée par le capteur (sans arrondi ni calibration).

#### temperature→get\_currentValue()

Retourne la valeur actuelle de la température.

#### temperature→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

#### temperature→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

#### temperature→get\_friendlyName()

Retourne un identifiant global du capteur de température au format NOM\_MODULE . NOM\_FONCTION.

#### temperature→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### temperature→get\_functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

#### temperature→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format SERIAL.FUNCTIONID.
<b>temperature→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>

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() yFindTemperature()yFindTemperature()

YTemperature

Permet de retrouver un capteur de température d'après un identifiant donné.

**function yFindTemperature( 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.

```
function 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()****YTemperature**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**function calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→describe()temperature.describe()****YTemperature**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **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 température (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**temperature→get\_advertisedValue()**  
**temperature→advertisedValue()**  
**temperature.get\_advertisedValue()**

**YTemperature**

---

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

```
function 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).

```
function 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()**  
**temperature→currentValue()**  
**temperature.get\_currentValue()**

---

**YTemperature**

Retourne la valeur actuelle de la température.

```
function 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()**  
**temperature→errorMessage()**  
**temperature.getErrorMessage()****YTemperature**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
function 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()**  
**temperature→errorType()**  
**temperature.get\_errorType()**

**YTemperature**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

**function 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()**  
**temperature→friendlyName()**  
**temperature.get\_friendlyName()**

**YTemperature**

Retourne un identifiant global du capteur de température au format NOM\_MODULE.NOM\_FONCTION.

```
function 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()**  
**temperature→functionDescriptor()**  
**temperature.get\_functionDescriptor()**

---

**YTemperature**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**temperature→get\_functionId()**  
**temperature→functionId()**  
**temperature.get\_functionId()**

**YTemperature**

Retourne l'identifiant matériel du capteur de température, sans référence au module.

```
function 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()**  
**temperature→hardwareId()**  
**temperature.get\_hardwareId()**

**YTemperature**

Retourne l'identifiant matériel unique du capteur de température au format SERIAL.FUNCTIONID.

**function 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()**  
**temperature→highestValue()**  
**temperature.get\_highestValue()**

**YTemperature**

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

```
function 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()**  
**temperature→logFrequency()**  
**temperature.get\_logFrequency()**

**YTemperature**

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.

**function 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()**  
**temperature→logicalName()**  
**temperature.get\_logicalName()**

**YTemperature**

Retourne le nom logique du capteur de température.

```
function 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()**  
**temperature→lowestValue()**  
**temperature.get\_lowestValue()**

**YTemperature**

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

**function get\_lowestValue( )**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_LOWESTVALUE\_INVALID**.

**temperature→get\_module()****YTemperature****temperature→module()temperature.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function 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>temperature→get_module_async()</b>	<b>YTemperature</b>
<b>temperature→module_async()</b>	
<b>temperature.get_module_async()</b>	

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

#### Paramètres :

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

#### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**temperature→get\_recordedData()**  
**temperature→recordedData()**  
**temperature.get\_recordedData()**

**YTemperature**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( startTime, endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**temperature→get\_reportFrequency()**  
**temperature→reportFrequency()**  
**temperature.get\_reportFrequency()**

**YTemperature**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**function get\_reportFrequency( )**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**temperature→get\_resolution()**  
**temperature→resolution()**  
**temperature.get\_resolution()**

YTemperature

Retourne la résolution des valeurs mesurées.

**function 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()**  
**temperature→sensorType()**  
**temperature.get\_sensorType()**

**YTemperature**

Retourne le type de capteur de température utilisé par le module

**function 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.

function **get\_unit( )**

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne **Y\_UNIT\_INVALID**.

**temperature→get(userData)**

**YTemperature**

**temperature→userData()temperature.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function 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()temperature.isOnline()****YTemperature**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()**  
**temperature.isOnline\_async()****YTemperature**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**temperature→load()temperature.load()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

function **load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→loadCalibrationPoints()**  
**temperature.loadCalibrationPoints()****YTemperature**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→load\_async()temperature.load\_async()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**temperature→nextTemperature()**  
**temperature.nextTemperature()**

---

**YTemperature**

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

```
function nextTemperature( )
```

**Retourne :**

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**temperature→registerTimedReportCallback()  
temperature.registerTimedReportCallback()****YTemperature**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**temperature→registerValueCallback()**  
**temperature.registerValueCallback()****YTemperature**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**temperature→setHighestValue()**  
**temperature.set\_highestValue()**

**YTemperature**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logFrequency()**  
**temperature→setLogFrequency()**  
**temperature.set\_logFrequency()**

**YTemperature**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logicalName()**  
**temperature→setLogicalName()**  
**temperature.set\_logicalName()**

**YTemperature**

Modifie le nom logique du capteur de température.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du capteur de température.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_lowestValue()**  
**temperature→setLowestValue()**  
**temperature.set\_lowestValue()**

**YTemperature**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_reportFrequency()**  
**temperature→setReportFrequency()**  
**temperature.set\_reportFrequency()**

**YTemperature**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval )**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_resolution()**  
**temperature→setResolution()**  
**temperature.set\_resolution()**

---

**YTemperature**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( newval)**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_sensorType()**  
**temperature→setSensorType()**  
**temperature.set\_sensorType()**

**YTemperature**

Change le type de senseur utilisé par le module.

```
function set_sensorType( newval )
```

Cette fonction sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital. N'oubliez pas d'appeler la méthode saveToFlash() du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi Y\_SENSORTYPE\_DIGITAL, Y\_SENSORTYPE\_TYPE\_K, Y\_SENSORTYPE\_TYPE\_E, Y\_SENSORTYPE\_TYPE\_J, Y\_SENSORTYPE\_TYPE\_N, Y\_SENSORTYPE\_TYPE\_R, Y\_SENSORTYPE\_TYPE\_S, Y\_SENSORTYPE\_TYPE\_T, Y\_SENSORTYPE\_PT100\_4WIRES, Y\_SENSORTYPE\_PT100\_3WIRES et Y\_SENSORTYPE\_PT100\_2WIRES

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set(userData)**  
**temperature→setUserData()**  
**temperature.set(userData)**

**YTemperature**

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).

```
function set(userData) {
```

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

**temperature→wait\_async()temperature.wait\_async()****YTemperature**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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 renournée 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.

<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.

### 3. Reference

---

**tilt→set\_logicalName(newval)**

Modifie le nom logique de l'inclinomètre.

**tilt→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**tilt→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**tilt→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**tilt→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**tilt→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YTilt.FindTilt()****YTilt****yFindTilt()yFindTilt()**

Permet de retrouver un inclinomètre d'après un identifiant donné.

```
function yFindTilt( 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() yFirstTilt()yFirstTilt()

YTilt

Commence l'énumération des inclinomètres accessibles par la librairie.

```
function 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.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→describe()tilt.describe()****YTilt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function 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'inclinomètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

---

**tilt→get\_advertisedValue()****YTilt****tilt→advertisedValue()tilt.get\_advertisedValue()**

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**tilt→get\_currentRawValue()** YTilt  
**tilt→currentRawValue()tilt.get\_currentRawValue()**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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.

```
function 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.

<b>tilt→getErrorMessage()</b>	<b>YTilt</b>
<b>tilt→errorMessage()tilt.getErrorMessage()</b>	

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
function 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'inclinomètre.

**tilt→get\_errorType()****YTilt****tilt→errorType()tilt.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
function 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.

---

<b>tilt→get_friendlyName()</b>	<b>YTilt</b>
<b>tilt→friendlyName()tilt.get_friendlyName()</b>	

---

Retourne un identifiant global de l'inclinomètre au format NOM\_MODULE . NOM\_FONCTION.

```
function 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()****YTilt****tilt→functionDescriptor()tilt.get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function 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>tilt→get_functionId()</b>	<b>YTilt</b>
<b>tilt→functionId()tilt.get_functionId()</b>	

---

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**tilt→get\_hardwareId()**

YTilt

**tilt→hardwareId()tilt.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'inclinomètre au format SERIAL.FUNCTIONID.

```
function 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.

<b>tilt→get_highestValue()</b>	<b>YTilt</b>
<b>tilt→highestValue()tilt.get_highestValue()</b>	

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

```
function get_highestValue( )
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**tilt→get\_logFrequency()****YTilt****tilt→logFrequency()tilt.get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function 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>tilt→get_logicalName()</b>	<b>YTilt</b>
<b>tilt→logicalName()tilt.get_logicalName()</b>	

Retourne le nom logique de l'inclinomètre.

```
function 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.

```
function 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**.

<b>tilt→get_module()</b>	<b>YTilt</b>
<b>tilt→module()tilt.get_module()</b>	

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**tilt→get\_module\_async()**

YTilt

**tilt→module\_async()tilt.get\_module\_async()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

<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.

**function get\_recordedData( startTime, endTime )**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**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.

```
function 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.

```
function 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.

```
function 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).

```
function 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()tilt.isOnline()****YTilt**

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()tilt.isOnline\_async()**

YTilt

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context )
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**tilt→load()tilt.load()****YTilt**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

function **load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→loadCalibrationPoints()  
tilt.loadCalibrationPoints()****YTilt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## tilt→load\_async()tilt.load\_async()

YTilt

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

### Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## **tilt→nextTilt()tilt.nextTilt()**

**YTilt**

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**tilt→registerValueCallback()  
tilt.registerValueCallback()****YTilt**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>tilt→set_logFrequency()</b>	<b>YTilt</b>
<b>tilt→setLogFrequency()tilt.set_logFrequency()</b>	

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_logicalName()**

YTilt

**tilt→setLogicalName()tilt.set\_logicalName()**

Modifie le nom logique de l'inclinomètre.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'inclinomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_lowestValue()** YTilt  
**tilt→setLowestValue()tilt.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_reportFrequency()**

YTilt

**tilt→setReportFrequency()tilt.set\_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_resolution()** YTilt  
**tilt→setResolution()tilt.set\_resolution()**

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set(userData)**

YTilt

**tilt→setUserData()tilt.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**tilt→wait\_async()tilt.wait\_async()**

YTilt

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.40. Interface de la fonction Voc

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_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.

**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)**

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() yFindVoc()yFindVoc()

YVoc

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

**function yFindVoc( 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.

```
function 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.

function **calibrateFromPoints( rawValues, refValues )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→describe()voc.describe()****YVoc**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**function describe( )**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex:  
Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**voc→get\_advertisedValue()** YVoc  
**voc→advertisedValue()voc.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

**function get\_advertisedValue( )**

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**voc→get\_currentRawValue()****YVoc****voc→currentRawValue()voc.get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**voc→get\_currentValue()**

**YVoc**

**voc→currentValue()voc.get\_currentValue()**

---

Retourne la mesure actuelle du taux de VOC estimé.

```
function 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.

```
function 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()  
voc→errorType()voc.get\_errorType()****YVoc**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Volatils.

**function 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 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.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voc→get\_functionId()****YVoc****voc→functionId()voc.get\_functionId()**

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

```
function 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.

**function 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.

```
function 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.

```
function 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.

```
function 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.

```
function 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.

```
function 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\_module\_async()** YVoc  
**voc→module\_async()voc.get\_module\_async()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**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.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voc→get\_reportFrequency()****YVoc****voc→reportFrequency()|voc.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function 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.

```
function 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.

**function 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).

```
function 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()voc.isOnline()****YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()|voc.isOnline\_async()****YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen  
**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**voc→load()voc.load()****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

**function load( msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→loadCalibrationPoints()**  
**voc.loadCalibrationPoints()****YVoc**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→load\_async()voc.load\_async()****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**voc→nextVoc()voc.nextVoc()****YVoc**

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function set_highestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour le taux de VOC estimé

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_logFrequency()****YVoc****voc→setLogFrequency()voc.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_logicalName()** YVoc  
**voc→setLogicalName()voc.set\_logicalName()**

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_lowestValue()****YVoc****voc→setLowestValue()voc.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

```
function set_lowestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour le taux de VOC estimé

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_reportFrequency()**  
**voc→setReportFrequency()**  
**voc.set\_reportFrequency()**

**YVoc**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_resolution()****YVoc****voc→setResolution()voc.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

function **set\_resolution( newval )**

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set(userData)**

YVoc

**voc→setUserData()|voc.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**voc→wait\_async()|voc.wait\_async()****YVoc**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.41. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_voltage.js'></script>
nodejs var yoctolib = require('yoctolib');
var YVoltage = yoctolib.YVoltage;
require_once('yocto_voltage.php');
#include "yocto_voltage.h"
m #import "yocto_voltage.h"
pas uses yocto_voltage;
vb yocto_voltage.vb
cs yocto_voltage.cs
java import com.yoctopuce.YoctoAPI.YVoltage;
py from yocto_voltage import *

```

### Fonction globales

#### **yFindVoltage(func)**

Permet de retrouver un capteur de tension d'après un identifiant donné.

#### **yFirstVoltage()**

Commence l'énumération des capteurs de tension accessibles par la librairie.

### Méthodes des objets YVoltage

#### **voltage→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **voltage→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **voltage→get\_advertisedValue()**

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

#### **voltage→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **voltage→get\_currentValue()**

Retourne la valeur instantanée de la tension.

#### **voltage→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### **voltage→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### **voltage→get\_friendlyName()**

Retourne un identifiant global du capteur de tension au format NOM\_MODULE . NOM\_FONCTION.

#### **voltage→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **voltage→get\_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

#### **voltage→get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.
<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>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**voltage→set\_logicalName(newval)**

Modifie le nom logique du capteur de tension.

**voltage→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la tension.

**voltage→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**voltage→set\_resolution(newval)**

Modifie la résolution des valeurs mesurées.

**voltage→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**voltage→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YVoltage.FindVoltage()

## YVoltage

### yFindVoltage()yFindVoltage()

Permet de retrouver un capteur de tension d'après un identifiant donné.

function **yFindVoltage( 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() yFirstVoltage()yFirstVoltage()**

**YVoltage**

Commence l'énumération des capteurs de tension accessibles par la librairie.

```
function 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.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**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.

```
function 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 tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**voltage→get\_advertisedValue()**  
**voltage→advertisedValue()**  
**voltage.get\_advertisedValue()**

**YVoltage**

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de tension (pas plus de 6 caractères).  
En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**voltage→get\_currentRawValue()**  
**voltage→currentRawValue()**  
**voltage.get\_currentRawValue()**

**YVoltage**

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function 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.

---

<b>voltage→get_currentValue()</b>	<b>YVoltage</b>
<b>voltage→currentValue()voltage.get_currentValue()</b>	

---

Retourne la valeur instantanée de la tension.

```
function get_currentValue( )
```

**Retourne :**

une valeur numérique représentant la valeur instantanée de la tension

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**voltage→get\_errorMessage()**

**YVoltage**

**voltage→errorMessage()voltage.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

**voltage→get\_errorType()****YVoltage****voltage→errorType()voltage.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function 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.

---

<b>voltage→get_friendlyName()</b>	<b>YVoltage</b>
<b>voltage→friendlyName()voltage.get_friendlyName()</b>	

---

Retourne un identifiant global du capteur de tension au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**voltage→get\_functionDescriptor()**  
**voltage→functionDescriptor()**  
**voltage.get\_functionDescriptor()**

**YVoltage**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function 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

**voltage→get\_functionId()**

**YVoltage**

**voltage→functionId()voltage.get\_functionId()**

---

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

function **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.

```
function 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.

**voltage→get\_highestValue()**

**YVoltage**

**voltage→highestValue()voltage.get\_highestValue()**

---

Retourne la valeur maximale observée pour la tension.

```
function 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.

```
function get_logFrequency( )
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**voltage→get\_logicalName()**

**YVoltage**

**voltage→logicalName()voltage.get\_logicalName()**

---

Retourne le nom logique du capteur de tension.

```
function 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.

```
function 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**voltage→get\_module\_async()**  
**voltage→module\_async()**  
**voltage.get\_module\_async()**

YVoltage

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module\_async( callback, context)**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

<b>voltage→get_recordedData()</b>	<b>YVoltage</b>
<b>voltage→recordedData()voltage.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.

**function get\_recordedData( startTime, endTime )**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voltage→get\_reportFrequency()**  
**voltage→reportFrequency()**  
**voltage.get\_reportFrequency()**

**YVoltage**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function 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()**  
**voltage→resolution()voltage.get\_resolution()**

---

**YVoltage**

Retourne la résolution des valeurs mesurées.

```
function 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.

```
function 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).

```
function get(userData) {
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**voltage→isOnline()voltage.isOnline()****YVoltage**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

**function 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→isOnline\_async()voltage.isOnline\_async()****YVoltage**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**voltage→load()voltage.load()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

function **load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→loadCalibrationPoints()****YVoltage****voltage.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
function loadCalibrationPoints( rawValues, refValues)
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→load\_async()voltage.load\_async()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## voltage→nextVoltage()voltage.nextVoltage()

YVoltage

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

```
function 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.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**voltage→registerValueCallback()  
voltage.registerValueCallback()****YVoltage**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**voltage→set\_highestValue()**  
**voltage→setHighestValue()**  
**voltage.set\_highestValue()**

**YVoltage**

Modifie la mémoire de valeur maximale observée pour la tension.

```
function set_highestValue( newval)
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée pour la tension

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_logFrequency()**  
**voltage→setLogFrequency()**  
**voltage.set\_logFrequency()**

**YVoltage**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>voltage→set_logicalName()</b>	<b>YVoltage</b>
<b>voltage→setLogicalName()voltage.set_logicalName()</b>	

---

Modifie le nom logique du capteur de tension.

```
function set_logicalName( newval )
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de tension.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_lowestValue()**

**YVoltage**

**voltage→setLowestValue()voltage.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée pour la tension.

```
function set_lowestValue( newval )
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour la tension

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_reportFrequency()**  
**voltage→setReportFrequency()**  
**voltage.set\_reportFrequency()**

YVoltage

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( newval)**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_resolution()** YVoltage  
**voltage→setResolution()voltage.set\_resolution()**

---

Modifie la résolution des valeurs mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set(userData)****YVoltage****voltage→setUserData()voltage.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**voltage→wait\_async()voltage.wait\_async()****YVoltage**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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 *

### Fonction globales

#### yFindVSource(func)

Permet de retrouver une source de tension d'après un identifiant donné.

#### yFirstVSource()

Commence l'énumération des sources de tension accessibles par la librairie.

### Méthodes des objets YVSource

#### vsource→describe()

Retourne un court texte décrivant la fonction au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### vsource→get\_advertisedValue()

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

#### vsource→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

#### vsource→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

#### vsource→get\_extPowerFailure()

Rend TRUE si le voltage de l'alimentation externe est trop bas.

#### vsource→get\_failure()

Indique si le module est en condition d'erreur.

#### vsource→get\_friendlyName()

Retourne un identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

#### vsource→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### vsource→get\_functionId()

Retourne l'identifiant matériel de la fonction, sans référence au module.

#### vsource→get\_hardwareId()

Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

#### vsource→get\_logicalName()

Retourne le nom logique de la source de tension.

#### vsource→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### vsource→get\_module\_async(callback, context)

### 3. Reference

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 rapport 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.FindVSource()yFindVSource()

Permet de retrouver une source de tension d'après un identifiant donné.

```
function yFindVSource( func)
```

## yFindVSource() — YVSource.FindVSource()yFindVSource()

Permet de retrouver une source de tension d'après un identifiant donné.

js	<code>function yFindVSource( func)</code>
php	<code>function yFindVSource( \$func)</code>
cpp	<code>YVSource* yFindVSource( const string&amp; func)</code>
m	<code>YVSource* yFindVSource( NSString* func)</code>
pas	<code>function yFindVSource( func: string): TYVSource</code>
vb	<code>function yFindVSource( ByVal func As String) As YVSource</code>
cs	<code>YVSource FindVSource( string func)</code>
java	<code>YVSource FindVSource( String func)</code>
py	<code>def FindVSource( func)</code>

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'ambiguité 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.

**function yFirstVSource( )**

**yFirstVSource() — YVSource.FirstVSource()yFirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

**js** function **yFirstVSource( )**

**php** function **yFirstVSource( )**

**cpp** YVSource\* **yFirstVSource( )**

**m** YVSource\* **yFirstVSource( )**

**pas** function **yFirstVSource( )**: TYVSource

**vb** function **yFirstVSource( )** As YVSource

**cs** YVSource **FirstVSource( )**

**java** YVSource **FirstVSource( )**

**py** def **FirstVSource( )**

Utiliser la fonction `YVSource.nextVSource()` pour itérer sur les autres sources de tension.

**Retourne :**

un pointeur sur un objet `YVSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de sources de tension disponibles.

**vsource→describe()vsource.describe()****YVSource**

Retourne un court texte décrivant la fonction au format TYPE ( NAME ) =SERIAL . FUNCTIONID.

**function describe( )**

**vsource→describe()vsource.describe()**

Retourne un court texte décrivant la fonction au format TYPE ( NAME ) =SERIAL . FUNCTIONID.

js	function <b>describe( )</b>
php	function <b>describe( )</b>
cpp	string <b>describe( )</b>
m	- <b>(NSString*) describe</b>
pas	function <b>describe( )</b> : string
vb	function <b>describe( )</b> As String
cs	string <b>describe( )</b>
java	String <b>describe( )</b>

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant la fonction (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**vsouce→get\_advertisedValue()**  
**vsouce→advertisedValue()**  
**vsouce.get\_advertisedValue()**

**YVSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

function **get\_advertisedValue( )**

**vsouce→get\_advertisedValue()**  
**vsouce→advertisedValue()vsouce.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**.

**vsource→getErrorMessage()**  
**vsource→errorMessage()**  
**vsource.getErrorMessage()**

**YVSource**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

function **getErrorMessage( )**

**vsource→getErrorMessage()**  
**vsource→errorMessage()vsource.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

**js** function **getErrorMessage( )**  
**php** function **getErrorMessage( )**  
**cpp** string **getErrorMessage( )**  
**m** -(NSString\*) errorMessage  
**pas** function **getErrorMessage( )**: string  
**vb** function **getErrorMessage( )** As String  
**cs** string **getErrorMessage( )**  
**java** String **getErrorMessage( )**  
**py** def **getErrorMessage( )**

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

---

<b>vsouce→get_errorType()</b>	<b>YVSource</b>
<b>vsouce→errorType()vsouce.get_errorType()</b>	

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
function get_errorType( )
```

---

<b>vsouce→get_errorType()</b>
<b>vsouce→errorType()vsouce.get_errorType()</b>

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
js function get_errorType( )
php function get_errorType( )
cpp YRETCODE get_errorType( )
pas function get_errorType( ): YRETCODE
vb function get_errorType( ) As YRETCODE
cs YRETCODE get_errorType( )
java int get_errorType( )
py def get_errorType( )
```

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

**vsources->get\_extPowerFailure()**  
**vsources->extPowerFailure()**  
**vsources.get\_extPowerFailure()**

**YVSource**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

function **get\_extPowerFailure( )**

**vsources->get\_extPowerFailure()**  
**vsources->extPowerFailure()vsources.get\_extPowerFailure()**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

<b>js</b>	function <b>get_extPowerFailure( )</b>
<b>php</b>	function <b>get_extPowerFailure( )</b>
<b>cpp</b>	<b>Y_EXTPOWERFAILURE_enum get_extPowerFailure( )</b>
<b>m</b>	-( <b>Y_EXTPOWERFAILURE_enum</b> ) extPowerFailure
<b>pas</b>	function <b>get_extPowerFailure( ): Integer</b>
<b>vb</b>	function <b>get_extPowerFailure( ) As Integer</b>
<b>cs</b>	<b>int get_extPowerFailure( )</b>
<b>java</b>	<b>int get_extPowerFailure( )</b>
<b>py</b>	<b>def get_extPowerFailure( )</b>
<b>cmd</b>	YVSource <b>target get_extPowerFailure</b>

**Retourne :**

soit **Y\_EXTPOWERFAILURE\_FALSE**, soit **Y\_EXTPOWERFAILURE\_TRUE**

En cas d'erreur, déclenche une exception ou retourne **Y\_EXTPOWERFAILURE\_INVALID**.

**vsouce→get\_failure()**  
**vsouce→failure()vsouce.get\_failure()**

YVSource

Indique si le module est en condition d'erreur.

```
function get_failure( )
```

**vsouce→get\_failure()**  
**vsouce→failure()vsouce.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()****YVSource****vsource→friendlyName()vsource.get\_friendlyName()**

Retourne un identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

```
function 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.

function **get\_functionDescriptor( )**

**vsource→get\_functionDescriptor()**  
**vsource→functionDescriptor()vsource.get\_vsourceDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
js function get_functionDescriptor( )
php function get_functionDescriptor( )
cpp YFUN_DESCR get_functionDescriptor( )
m -(YFUN_DESCR) functionDescriptor
pas function get_functionDescriptor( ): YFUN_DESCR
vb function get_functionDescriptor( ) As YFUN_DESCR
cs YFUN_DESCR get_functionDescriptor( )
java String get_functionDescriptor( )
py def get_functionDescriptor( )
```

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

**vsouce→get\_functionId()****YVSource****vsouce→functionId()vsouce.get\_vsourceld()**

Retourne l'identifiant matériel de la fonction, sans référence au module.

```
function get_functionId( )
```

**vsouce→get\_functionId()****vsouce→functionId()vsouce.get\_vsourceld()**

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.

**vsource→get\_hardwareId()****YVSource****vsource→hardwareId()vsource.get\_hardwareId()**

Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

function **get\_hardwareId( )**

**vsource→get\_hardwareId()****vsource→hardwareId()vsource.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.

**vsource→get\_logicalName()****YVSource****vsource→logicalName()vsource.get\_logicalName()**

Retourne le nom logique de la source de tension.

```
function get_logicalName( )
```

**vsource→get\_logicalName()****vsource→logicalName()vsource.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()**  
**vsource→module()vsource.get\_module()**

**YVSource**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **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 retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**vsource→get\_module\_async()**  
**vsource→module\_async()**  
**vsource.get\_module\_async()**

**YVSource**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**vsouce→get\_overCurrent()** **YVSource**  
**vsouce→overCurrent()vsouce.get\_overCurrent()**

---

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
function get_overCurrent( )
```

**vsouce→get\_overCurrent()**  
**vsouce→overCurrent()vsouce.get\_overCurrent()**

---

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.

```
function 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.

```
function 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.

function **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>	function <b>get_regulationFailure( )</b>
<b>php</b>	function <b>get_regulationFailure( )</b>
<b>cpp</b>	Y_REGULATIONFAILURE_enum <b>get_regulationFailure( )</b>
<b>m</b>	-(Y_REGULATIONFAILURE_enum) regulationFailure
<b>pas</b>	function <b>get_regulationFailure( )</b> : Integer
<b>vb</b>	function <b>get_regulationFailure( )</b> As Integer
<b>cs</b>	int <b>get_regulationFailure( )</b>
<b>java</b>	int <b>get_regulationFailure( )</b>
<b>py</b>	def <b>get_regulationFailure( )</b>
<b>cmd</b>	YVSource <b>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()****YVSource****vsOURCE→unit()vsOURCE.get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
function 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`.

**vsource→get(userData)****YVSource****vsource→userData()vsource.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) {
```

**vsource→get(userData)****vsource→userData()vsource.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.

**vsouce→get\_voltage()**  
**vsouce→voltage()vsouce.get\_voltage()**

**YVSource**

Retourne la valeur de la commande de tension de sortie en mV

```
function get_voltage( )
```

**vsouce→get\_voltage()**  
**vsouce→voltage()vsouce.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()|vsource.isOnline()****YVSource**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

**vsource→isOnline()|vsource.isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

js	function isOnline( )
php	function isOnline( )
cpp	bool isOnline( )
m	-(BOOL) isOnline
pas	function isOnline( ): boolean
vb	function isOnline( ) As Boolean
cs	bool isOnline( )
java	boolean isOnline( )
py	def isOnline( )

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si la fonction est joignable, false sinon

**vsource→isOnline\_async()vsource.isOnline\_async()****YVSource**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

**function isOnline\_async( callback, context)**

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**vsource→load()vsource.load()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**function load( msValidity)**

**vsource→load()vsource.load()**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

<b>js</b>	<b>function load( msValidity)</b>
<b>php</b>	<b>function load( \$msValidity)</b>
<b>cpp</b>	<b>YRETCODE load( int msValidity)</b>
<b>m</b>	<b>-(YRETCODE) load : (int) msValidity</b>
<b>pas</b>	<b>function load( msValidity: integer): YRETCODE</b>
<b>vb</b>	<b>function load( ByVal msValidity As Integer) As YRETCODE</b>
<b>cs</b>	<b>YRETCODE load( int msValidity)</b>
<b>java</b>	<b>int load( long msValidity)</b>
<b>py</b>	<b>def 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→load\_async()|vsource.load\_async()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**function load\_async( msValidity, callback, context)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**vsource→nextVSource()vsource.nextVSource()****YVSource**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

function **nextVSource()**

**vsource→nextVSource()vsource.nextVSource()**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

<code>js</code>	function <b>nextVSource()</b>
<code>php</code>	function <b>nextVSource()</b>
<code>cpp</code>	<b>YVSource * nextVSource()</b>
<code>m</code>	<b>-(YVSource*) nextVSource</b>
<code>pas</code>	function <b>nextVSource()</b> : TYVSource
<code>vb</code>	function <b>nextVSource()</b> As YVSource
<code>cs</code>	<b>YVSource nextVSource()</b>
<code>java</code>	<b>YVSource nextVSource()</b>
<code>py</code>	<b>def nextVSource()</b>

**Retourne :**

un pointeur sur un objet `YVSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**vsOURCE→pulse()vsOURCE.pulse()****YVSource**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
function pulse( voltage, ms_duration)
```

**vsOURCE→pulse()vsOURCE.pulse()**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
js function pulse( voltage, ms_duration)
php function pulse( $voltage, $ms_duration)
cpp int pulse( int voltage, int ms_duration)
m -(int) pulse : (int) voltage : (int) ms_duration
pas function pulse( voltage: integer, ms_duration: integer): integer
vb function pulse( ByVal voltage As Integer,
                  ByVal ms_duration As Integer) As Integer
cs int pulse( int voltage, int ms_duration)
java int pulse( int voltage, int ms_duration)
py def pulse( voltage, ms_duration)
cmd YVSource target pulse voltage ms_duration
```

**Paramètres :**

<b>voltage</b>	tension demandée, en millivolts
<b>ms_duration</b>	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.

```
function registerValueCallback( callback)
```

## **vsource→registerValueCallback()vsource.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback( callback)
php	function registerValueCallback( \$callback)
cpp	void registerValueCallback( YDisplayUpdateCallback callback)
pas	procedure registerValueCallback( callback: TGenericUpdateCallback)
vb	procedure registerValueCallback( ByVal callback As GenericUpdateCallback)
cs	void registerValueCallback( UpdateCallback callback)
java	void registerValueCallback( UpdateCallback callback)
py	def registerValueCallback( callback)
m	-(void) registerValueCallback : (YFunctionUpdateCallback) callback

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>vsouce→set_logicalName()</b>	<b>YVSource</b>
<b>vsouce→setLogicalName()</b>	
<b>vsouce.set_logicalName()</b>	

---

Modifie le nom logique de la source de tension.

function **set\_logicalName( newval)**

---

<b>vsouce→set_logicalName()</b>
<b>vsouce→setLogicalName()vsouce.set_logicalName()</b>

---

Modifie le nom logique de la source de tension.

```
js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YVSource target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName( )` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé.

#### Paramètres :

**newval** une chaîne de caractères représentant le nom logique de la source de tension

#### Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**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).

```
function set(userData( 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).

```
function set_voltage( 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.

```
function voltageMove( target, ms_duration )
```

**vsource→voltageMove()vsource.voltageMove()**

Déclenche une variation constante de la sortie vers une valeur donnée.

```
js   function voltageMove( target, ms_duration )
php  function voltageMove( $target, $ms_duration )
cpp  int voltageMove( int target, int ms_duration )
m    -(int) voltageMove : (int) target : (int) ms_duration
pas   function voltageMove( target: integer, ms_duration: integer): integer
vb    function voltageMove( ByVal target As Integer,
                    ByVal ms_duration As Integer) As Integer
cs    int voltageMove( int target, int ms_duration )
java  int voltageMove( int target, int ms_duration )
py    def voltageMove( target, ms_duration )
cmd   YVSource target voltageMove target ms_duration
```

**Paramètres :**

**target** nouvelle valeur de sortie à la fin de la transition, en millivolts.

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsource→wait\_async()vsource.wait\_async()****YVSource**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la VM Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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.

### 3. Reference

<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 yFirstWakeUpMonitor( ).
<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>

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.

```
function yFindWakeUpMonitor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.isOnline()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le moniteur sans ambiguïté

### Retourne :

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

## YWakeUpMonitor.FirstWakeUpMonitor() yFirstWakeUpMonitor()yFirstWakeUpMonitor()

## YWakeUpMonitor

Commence l'énumération des Moniteurs accessibles par la librairie.

```
function 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.

**function 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 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).

```
function get_advertisedValue( )
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

**wakeupmonitor→get\_errorMessage()**  
**wakeupmonitor→errorMessage()**  
**wakeupmonitor.get\_errorMessage()**

**YWakeUpMonitor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

**function get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

**wakeupmonitor→get\_errorType()**  
**wakeupmonitor→errorType()**  
**wakeupmonitor.get\_errorType()****YWakeUpMonitor**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

**function 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()**  
**wakeupmonitor→friendlyName()**  
**wakeupmonitor.get\_friendlyName()**

---

**YWakeUpMonitor**

Retourne un identifiant global du moniteur au format NOM\_MODULE.NOM\_FONCTION.

**function 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()**  
**wakeupmonitor→functionDescriptor()**  
**wakeupmonitor.get\_functionDescriptor()**

**YWakeUpMonitor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function 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

**wakeupmonitor→get\_functionId()**  
**wakeupmonitor→functionId()**  
**wakeupmonitor.get\_functionId()**

---

**YWakeUpMonitor**

Retourne l'identifiant matériel du moniteur, sans référence au module.

**function get\_functionId( )**

Par example `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()  
wakeupmonitor→hardwareId()  
wakeupmonitor.get\_hardwareId()

YWakeUpMonitor

Retourne l'identifiant matériel unique du moniteur au format SERIAL.FUNCTIONID.

```
function 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()  
wakeupmonitor→logicalName()  
wakeupmonitor.get\_logicalName()

YWakeUpMonitor

Retourne le nom logique du moniteur.

```
function 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()**  
**wakeupmonitor→module()**  
**wakeupmonitor.get\_module()****YWakeUpMonitor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function 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\_module\_async()**  
**wakeupmonitor→module\_async()**  
**wakeupmonitor.get\_module\_async()****YWakeUpMonitor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupmonitor→get\_nextWakeUp()

YWakeUpMonitor

wakeupmonitor→nextWakeUp()

wakeupmonitor.get\_nextWakeUp()

---

Retourne la prochaine date/heure de réveil agendée (format UNIX)

```
function 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()**  
**wakeupmonitor→powerDuration()**  
**wakeupmonitor.get\_powerDuration()**

**YWakeUpMonitor**

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

**function 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()**  
**wakeupmonitor→sleepCountdown()**  
**wakeupmonitor.get\_sleepCountdown()**

**YWakeUpMonitor**

Retourne le temps avant le prochain sommeil.

```
function 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)**  
**wakeupmonitor→userData()**  
**wakeupmonitor.get(userData)**

**YWakeUpMonitor**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function 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()**  
**wakeupmonitor→wakeUpReason()**  
**wakeupmonitor.get\_wakeUpReason()**

**YWakeUpMonitor**

Renvoie la raison du dernier réveil.

```
function 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()  
wakeupmonitor→wakeUpState()  
wakeupmonitor.get\_wakeUpState()

YWakeUpMonitor

Revoie l'état actuel du moniteur

```
function 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.

```
function 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→isOnline\_async()  
wakeupmonitor.isOnline\_async()****YWakeUpMonitor**

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**wakeupmonitor→load()wakeupmonitor.load()****YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

function **load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→load\_async()**  
**wakeupmonitor.load\_async()****YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**wakeupmonitor→nextWakeUpMonitor()**  
**wakeupmonitor.nextWakeUpMonitor()****YWakeUpMonitor**

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

```
function nextWakeUpMonitor( )
```

**Retourne :**

un pointeur sur un objet `YWakeUpMonitor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wakeupmonitor→registerValueCallback()**  
**wakeupmonitor.registerValueCallback()****YWakeUpMonitor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()**  
**wakeupmonitor.resetSleepCountDown()****YWakeUpMonitor**

Réinitialise le compteur de mise en sommeil.

```
function resetSleepCountDown( )
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set\_logicalName()  
wakeupmonitor→setLogicalName()  
wakeupmonitor.set\_logicalName()

YWakeUpMonitor

Modifie le nom logique du moniteur.

function **set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du moniteur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set\_nextWakeUp()**  
wakeupmonitor→**setNextWakeUp()**  
**wakeupmonitor.set\_nextWakeUp()**

**YWakeUpMonitor**

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
function set_nextWakeUp( newval )
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set\_powerDuration()  
wakeupmonitor→setPowerDuration()  
wakeupmonitor.set\_powerDuration()

YWakeUpMonitor

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

function **set\_powerDuration( newval )**

**Paramètres :**

**newval** un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set\_sleepCountdown()  
wakeupmonitor→setSleepCountdown()  
wakeupmonitor.set\_sleepCountdown()

YWakeUpMonitor

Modifie le temps avant le prochain sommeil .

```
function set_sleepCountdown( newval)
```

**Paramètres :**

**newval** un entier représentant le temps avant le prochain sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set(userData)  
wakeupmonitor→setUserData()  
wakeupmonitor.set(userData)

YWakeUpMonitor

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).

```
function set(userData) {
```

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.

function **sleep( secBeforeSleep )**

**Paramètres :**

**secBeforeSleep** nombre de seconde avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→sleepFor()  
wakeupmonitor.sleepFor()****YWakeUpMonitor**

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

**function sleepFor( secUntilWakeUp, secBeforeSleep)**

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

**Paramètres :****secUntilWakeUp** durée de la mise en sommeil, en secondes**secBeforeSleep** nombre de secondes avant la mise en sommeil**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→sleepUntil()**  
**wakeupmonitor.sleepUntil()****YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
function sleepUntil( wakeUpTime, secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

**Paramètres :**

**wakeUpTime** date/heure du réveil (format UNIX)

**secBeforeSleep** nombre de secondes avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→wait\_async()**  
**wakeupmonitor.wait\_async()****YWakeUpMonitor**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

**wakeupmonitor→wakeupmonitor.wakeUp()****YWakeUpMonitor**

Force un réveil.

```
function 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;
require_once('yocto_wakeupschedule.php');
#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()

Retourne les minutes de l'intervalle 00-29 de chaque heure où le réveil est actif.
<b>wakeupschedule→get_minutesB()</b>
Retourne les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif.
<b>wakeupschedule→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>wakeupschedule→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>wakeupschedule→get_monthDays()</b>
Retourne les jours du mois où le réveil est actif..
<b>wakeupschedule→get_months()</b>
Retourne les mois où le réveil est actif..
<b>wakeupschedule→get_nextOccurrence()</b>
Retourne la date/heure de la prochaine occurrence de réveil
<b>wakeupschedule→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>wakeupschedule→get_weekDays()</b>
Retourne les jours de la semaine où le réveil est actif..
<b>wakeupschedule→isOnline()</b>
Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.
<b>wakeupschedule→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.
<b>wakeupschedule→load(msValidity)</b>
Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.
<b>wakeupschedule→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.
<b>wakeupschedule→nextWakeUpSchedule()</b>
Continue l'énumération des réveils agendés commencée à l'aide de yFirstWakeUpSchedule( ).
<b>wakeupschedule→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>wakeupschedule→set_hours(newval, newval)</b>
Modifie les heures où un réveil doit avoir lieu
<b>wakeupschedule→set_logicalName(newval)</b>
Modifie le nom logique du réveil agendé.
<b>wakeupschedule→set_minutes(bitmap)</b>
Modifie toutes les minutes où un réveil doit avoir lieu
<b>wakeupschedule→set_minutesA(newval, newval)</b>
Modifie les minutes de l'intervalle 00-29 où un réveil doit avoir lieu
<b>wakeupschedule→set_minutesB(newval)</b>
Modifie les minutes de l'intervalle 30-59 où un réveil doit avoir lieu.
<b>wakeupschedule→set_monthDays(newval, newval)</b>
Modifie les jours du mois où un réveil doit avoir lieu
<b>wakeupschedule→set_months(newval, newval)</b>
Modifie les mois où un réveil doit avoir lieu
<b>wakeupschedule→set_userData(data)</b>

### **3. Reference**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **wakeupschedule→set\_weekDays(newval, newval)**

Modifie les jours de la semaine où un réveil doit avoir lieu

#### **wakeupschedule→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YWakeUpSchedule.FindWakeUpSchedule()****yFindWakeUpSchedule()yFindWakeUpSchedule()****YWakeUpSchedule**

Permet de retrouver un réveil agendé d'après un identifiant donné.

```
function yFindWakeUpSchedule( 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.

```
function 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.

```
function 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()

YWakeUpSchedule

wakeupschedule→advertisedValue()

wakeupschedule.get\_advertisedValue()

---

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

```
function 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()**  
**wakeupschedule→errorMessage()**  
**wakeupschedule.get\_errorMessage()****YWakeUpSchedule**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
function 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()**  
**wakeupschedule→errorType()**  
**wakeupschedule.get\_errorType()**

**YWakeUpSchedule**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
function 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()**  
**wakeupschedule→friendlyName()**  
**wakeupschedule.get\_friendlyName()****YWakeUpSchedule**

Retourne un identifiant global du réveil agendé au format NOM\_MODULE.NOM\_FONCTION.

```
function 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()**  
**wakeupschedule→functionDescriptor()**  
**wakeupschedule.get\_functionDescriptor()****YWakeUpSchedule**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**wakeupschedule→get\_functionId()**  
**wakeupschedule→functionId()**  
**wakeupschedule.get\_functionId()**

**YWakeUpSchedule**

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

function **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()  
wakeupschedule→hardwareId()  
wakeupschedule.get.hardwareId()

YWakeUpSchedule

Retourne l'identifiant matériel unique du réveil agendé au format SERIAL.FUNCTIONID.

```
function 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()  
wakeupschedule→hours()  
wakeupschedule.get\_hours()

YWakeUpSchedule

Retourne les heures où le réveil est actif..

```
function 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()**  
**wakeupschedule→logicalName()**  
**wakeupschedule.get\_logicalName()**

---

**YWakeUpSchedule**

Retourne le nom logique du réveil agendé.

```
function 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()**  
**wakeupschedule→minutes()**  
**wakeupschedule.get\_minutes()**

**YWakeUpSchedule**

Retourne toutes les minutes de chaque heure où le réveil est actif.

```
function get_minutes( )
```

**wakeupschedule→get\_minutesA()**  
**wakeupschedule→minutesA()**  
**wakeupschedule.get\_minutesA()**

**YWakeUpSchedule**

---

Retourne les minutes de l'intervalle 00-29 de chaque heures où le réveil est actif.

**function 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()  
wakeupschedule→minutesB()  
wakeupschedule.get\_minutesB()

YWakeUpSchedule

Retourne les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif.

```
function 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()**  
**wakeupschedule→module()**  
**wakeupschedule.get\_module()**

---

**YWakeUpSchedule**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**wakeupschedule→get\_module\_async()**  
**wakeupschedule→module\_async()**  
**wakeupschedule.get\_module\_async()****YWakeUpSchedule**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**wakeupschedule→get\_monthDays()**  
**wakeupschedule→monthDays()**  
**wakeupschedule.get\_monthDays()**

**YWakeUpSchedule**

Retourne les jours du mois où le réveil est actif..

```
function 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()**  
**wakeupschedule→months()**  
**wakeupschedule.get\_months()**

**YWakeUpSchedule**

Retourne les mois où le réveil est actif..

```
function 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

```
function 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)**  
**wakeupschedule→userData()**  
**wakeupschedule.get(userData)**

**YWakeUpSchedule**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function 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..

```
function 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.

function **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→isOnline\_async()**  
**wakeupschedule.isOnline\_async()****YWakeUpSchedule**

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit

trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**wakeupschedule→load()wakeupschedule.load()****YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

function **load( msValidity)**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→load\_async()  
wakeupschedule.load\_async()****YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

**wakeupschedule→nextWakeUpSchedule()**  
**wakeupschedule.nextWakeUpSchedule()****YWakeUpSchedule**

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

```
function nextWakeUpSchedule( )
```

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wakeupschedule→registerValueCallback()  
wakeupschedule.registerValueCallback()****YWakeUpSchedule**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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()  
wakeupschedule→setHours()  
wakeupschedule.set\_hours()

YWakeUpSchedule

Modifie les heures où un réveil doit avoir lieu

```
function set_hours( newval)
```

**Paramètres :**

**newval** un entier représentant les heures où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_logicalName()**  
**wakeupschedule→setLogicalName()**  
**wakeupschedule.set\_logicalName()**

**YWakeUpSchedule**

Modifie le nom logique du réveil agendé.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du réveil agendé.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_minutes()  
wakeupschedule→setMinutes()  
wakeupschedule.set\_minutes()

YWakeUpSchedule

Modifie toutes les minutes où un réveil doit avoir lieu

function **set\_minutes( bitmap )**

**Paramètres :**

**bitmap** Minutes 00-59 de chaque heure où le réveil est actif.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_minutesA()**  
**wakeupschedule→setMinutesA()**  
**wakeupschedule.set\_minutesA()**

**YWakeUpSchedule**

Modifie les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

function **set\_minutesA( 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()**  
**wakeupschedule→setMinutesB()**  
**wakeupschedule.set\_minutesB()**

**YWakeUpSchedule**

Modifie les minutes de l'intervalle 30-59 où un réveil doit avoir lieu.

function **set\_minutesB( 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()  
wakeupschedule→setMonthDays()  
wakeupschedule.set\_monthDays()

YWakeUpSchedule

Modifie les jours du mois où un réveil doit avoir lieu

function **set\_monthDays( newval )**

**Paramètres :**

**newval** un entier représentant les jours du mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_months()  
wakeupschedule→setMonths()  
wakeupschedule.set\_months()

YWakeUpSchedule

Modifie les mois où un réveil doit avoir lieu

```
function set_months( newval)
```

**Paramètres :**

**newval** un entier représentant les mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set(userData)  
wakeupschedule→setUserData()  
wakeupschedule.set(userData)

YWakeUpSchedule

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).

```
function set(userData) {
```

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()**  
**wakeupschedule→setWeekDays()**  
**wakeupschedule.set\_weekDays()**

**YWakeUpSchedule**

Modifie les jours de la semaine où un réveil doit avoir lieu

```
function set_weekDays( newval)
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→wait\_async()  
wakeupschedule.wait\_async()****YWakeUpSchedule**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**function wait\_async( callback, context)**

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 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>
node.js	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.
<b>watchdog→get_hardwareId()</b>
Retourne l'identifiant matériel unique du watchdog au format SERIAL . FUNCTIONID.
<b>watchdog→get_logicalName()</b>
Retourne le nom logique du watchdog.
<b>watchdog→get_maxTimeOnStateA()</b>
Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.
<b>watchdog→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.
<b>watchdog→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>watchdog→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>watchdog→get_output()</b>
Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.
<b>watchdog→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>watchdog→get_running()</b>
Retourne l'état du watchdog.
<b>watchdog→get_state()</b>
Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).
<b>watchdog→get_stateAtPowerOn()</b>
Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).
<b>watchdog→get_triggerDelay()</b>
Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.
<b>watchdog→get_triggerDuration()</b>
Retourne la durée d'un reset généré par le watchdog, en millisecondes.
<b>watchdog→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>watchdog→isOnline()</b>
Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.
<b>watchdog→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.
<b>watchdog→load(msValidity)</b>
Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.
<b>watchdog→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.
<b>watchdog→nextWatchdog()</b>
Continue l'énumération des watchdog commencée à l'aide de yFirstWatchdog( ).
<b>watchdog→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).

**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.

```
function yFindWatchdog( 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.

```
function 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

```
function delayedPulse( ms_delay, ms_duration )
```

**Paramètres :**

**ms\_delay**      délai d'attente avant l'impulsion, en millisecondes

**ms\_duration**      durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→describe()watchdog.describe()****YWatchdog**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function 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 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).

```
function 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.

```
function get_autoStart( )
```

**Retourne :**

soit Y\_AUTOSTART\_OFF, soit Y\_AUTOSTART\_ON, selon l'état du watchdog à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_AUTOSTART\_INVALID.

**watchdog→get\_countdown()**

**YWatchdog**

**watchdog→countdown()watchdog.get\_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

```
function 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.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

### **watchdog→get\_errorType()**

**YWatchdog**

### **watchdog→errorType()watchdog.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
function 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.

```
function 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.

**watchdog→get\_functionDescriptor()**  
**watchdog→functionDescriptor()**  
**watchdog.get\_functionDescriptor()**

---

**YWatchdog**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**watchdog→get\_functionId()****YWatchdog****watchdog→functionId()watchdog.get\_functionId()**

Retourne l'identifiant matériel du watchdog, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**watchdog→get\_hardwareId()**

**YWatchdog**

**watchdog→hardwareId()watchdog.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du watchdog au format SERIAL.FUNCTIONID.

```
function 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.

```
function 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()**  
**watchdog→maxTimeOnStateA()**  
**watchdog.get\_maxTimeOnStateA()**

**YWatchdog**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**function get\_maxTimeOnStateA( )**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEA\_INVALID.

**watchdog→get\_maxTimeOnStateB()****YWatchdog****watchdog→maxTimeOnStateB()****watchdog.get\_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
function 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**watchdog→get\_module\_async()**  
**watchdog→module\_async()**  
**watchdog.get\_module\_async()****YWatchdog**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**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.

```
function get_output( )
```

**Retourne :**

soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUT_INVALID`.

**watchdog→get\_pulseTimer()****YWatchdog****watchdog→pulseTimer()watchdog.get\_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
function 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.

```
function 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).

```
function 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()**  
**watchdog→stateAtPowerOn()**  
**watchdog.get\_stateAtPowerOn()**

**YWatchdog**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

function **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.

```
function 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.

```
function 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).

```
function 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()watchdog.isOnline()

YWatchdog

---

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
function 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→isOnline\_async()  
watchdog.isOnline\_async()****YWatchdog**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**watchdog→load()watchdog.load()****YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→load\_async()|watchdog.load\_async()****YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**watchdog→nextWatchdog()**  
**watchdog.nextWatchdog()**

---

**YWatchdog**

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

```
function nextWatchdog( )
```

**Retourne :**

un pointeur sur un objet `YWatchdog` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**watchdog→pulse()watchdog.pulse()****YWatchdog**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
function pulse( ms_duration)
```

**Paramètres :**

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→registerValueCallback()  
watchdog.registerValueCallback()****YWatchdog**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
function 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.

```
function set_autoStart( newval )
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

### Paramètres :

**newval** soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watching au démarrage du module

### Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_logicalName()  
watchdog→setLogicalName()  
watchdog.set\_logicalName()****YWatchdog**

Modifie le nom logique du watchdog.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du watchdog.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_maxTimeOnStateA()**  
**watchdog→setMaxTimeOnStateA()**  
**watchdog.set\_maxTimeOnStateA()**

**YWatchdog**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

function **set\_maxTimeOnStateA( newval )**

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_maxTimeOnStateB()**  
**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.

```
function set_maxTimeOnStateB( newval)
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **watchdog→set\_output()**

**YWatchdog**

## **watchdog→setOutput()watchdog.set\_output()**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
function set_output( newval )
```

### **Paramètres :**

**newval** soit **Y\_OUTPUT\_OFF**, soit **Y\_OUTPUT\_ON**, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

### **Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_running()****YWatchdog****watchdog→setRunning()watchdog.set\_running()**

Modifie manuellement l'état de fonctionnement du watchdog.

```
function set_running( newval)
```

**Paramètres :**

**newval** soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon manuellement l'état de fonctionnement du watchdog

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **watchdog→set\_state()**

**YWatchdog**

## **watchdog→setState()watchdog.set\_state()**

---

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
function set_state( newval )
```

### **Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_stateAtPowerOn()**  
**watchdog→setStateAtPowerOn()**  
**watchdog.set\_stateAtPowerOn()**

**YWatchdog**

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).

```
function set_stateAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDelay()**  
**watchdog→setTriggerDelay()**  
**watchdog.set\_triggerDelay()**

**YWatchdog**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

```
function set_triggerDelay( newval )
```

**Paramètres :**

**newval** un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDuration()**  
**watchdog→setTriggerDuration()**  
**watchdog.set\_triggerDuration()**

**YWatchdog**

Modifie la durée des resets générés par le watchdog, en millisecondes.

```
function set_triggerDuration( newval )
```

**Paramètres :**

**newval** un entier représentant la durée des resets générés par le watchdog, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set(userData)**

**YWatchdog**

**watchdog→setUserData()|watchdog.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**watchdog→wait\_async()watchdog.wait\_async()****YWatchdog**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

## 3.46. Interface de la fonction Wireless

La fonction YWireless permet de configurer et de contrôler la configuration du réseau sans fil sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_wireless.js'></script>
nodejs var yoctolib = require('yoctolib');
var YWireless = yoctolib.YWireless;
require_once('yocto_wireless.php');
#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()

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(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### wireless→wait\_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YWireless.FindWireless() yFindWireless()yFindWireless()

YWireless

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

**function yFindWireless( 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.

```
function 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".

**function adhocNetwork( ssid, securityKey)**

Si une clef d'accès est spécifiée, le réseau sera protégé par une sécurité WEP128 (l'utilisation de WPA n'est pas standardisée en mode ad-hoc). N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :****ssid** nom du réseau sans fil à créer**securityKey** clé d'accès de réseau, sous forme de chaîne de caractères**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→describe()wireless.describe()****YWireless**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE ( NAME ) =SERIAL.FUNCTIONID.

function **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).

```
function 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.

```
function 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()</b>	
<b>wireless.get_detectedWlans()</b>	

---

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

```
function 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 retournée.

**Retourne :**

une liste d'objets YWlanRecord, contenant le SSID, le canal, la qualité du signal, et l'algorithme de sécurité utilisé par le réseau sans-fil

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**wireless→get\_errorMessage()**  
**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.

```
function 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.

**function get\_errorType( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

**wireless→get\_friendlyName()****YWireless****wireless→friendlyName()wireless.get\_friendlyName()**

Retourne un identifiant global de l'interface réseau sans fil au format NOM\_MODULE.NOM\_FONCTION.

```
function 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.

**function get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**wireless→get\_functionId()****YWireless****wireless→functionId()wireless.get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

```
function 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\_hwrid()**

**YWireless**

**wireless→hwrid()wireless.get\_hwrid()**

---

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

**function get\_hwrid( )**

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.

```
function 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()**

**YWireless**

**wireless→logicalName()wireless.get\_logicalName()**

---

Retourne le nom logique de l'interface réseau sans fil.

```
function 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 diagnostique de l'interface au réseau sans fil.

```
function get_message( )
```

**Retourne :**

une chaîne de caractères représentant le dernier message de diagnostique 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.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**wireless→get\_module\_async()**  
**wireless→module\_async()**  
**wireless.get\_module\_async()**

**YWireless**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module\_async( callback, context)**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**wireless→get\_security()**

**YWireless**

**wireless→security()wireless.get\_security()**

---

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

```
function 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.

```
function 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).

```
function 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.

**function 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→isOnline\_async()wireless.isOnline\_async()****YWireless**

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

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.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**wireless→joinNetwork()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").

function **joinNetwork( ssid, securityKey)**

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à utiliser

**securityKey** clé d'accès au réseau, sous forme de chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→load()|wireless.load()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

**function load( msValidity )**

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→load\_async()wireless.load\_async()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI\_SUCCESS)

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

## wireless→nextWireless()**wireless.nextWireless()**

**YWireless**

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

```
function 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.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

**wireless→set\_logicalName()**  
**wireless→setLogicalName()**  
**wireless.set\_logicalName()**

**YWireless**

Modifie le nom logique de l'interface réseau sans fil.

**function set\_logicalName( newval )**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→set(userData)****YWireless****wireless→setUserData()|wireless.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
function set(userData) {
```

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

**wireless→wait\_async()|wireless.wait\_async()****YWireless**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

**Paramètres :**

**callback** fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

**context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

**Retourne :**

rien du tout :

# Index

## A

Accelerometer 32  
adhocNetwork, YWireless 1723  
Alimentation 487  
AnButton 78

## B

Blueprint 12  
Brute 341

## C

calibrate, YLightSensor 761  
calibrateFromPoints, YAccelerometer 36  
calibrateFromPoints, YCarbonDioxide 124  
calibrateFromPoints, YCompass 200  
calibrateFromPoints, YCurrent 244  
calibrateFromPoints, YGenericSensor 553  
calibrateFromPoints, YGyro 603  
calibrateFromPoints, YHumidity 687  
calibrateFromPoints, YLightSensor 762  
calibrateFromPoints, YMagnetometer 805  
calibrateFromPoints, YPower 990  
calibrateFromPoints, YPressure 1037  
calibrateFromPoints, YQt 1149  
calibrateFromPoints,YSensor 1303  
calibrateFromPoints, YTemperature 1385  
calibrateFromPoints, YTilt 1430  
calibrateFromPoints, YVoc 1473  
calibrateFromPoints, YVoltage 1516  
callbackLogin, YNetwork 903  
cancel3DCalibration, YRefFrame 1223  
CarbonDioxide 120  
CheckLogicalName, YAPI 14  
clear, YDisplayLayer 456  
clearConsole, YDisplayLayer 457  
ColorLed 163  
Compass 196  
Configuration 1219  
consoleOut, YDisplayLayer 458  
Contrôle 3, 5, 487, 853, 959  
copyLayerContent, YDisplay 408  
Current 240

## D

DataLogger 283  
delayedPulse, YDigitalIO 360  
delayedPulse, YRelay 1263  
delayedPulse, YWatchdog 1675  
describe, YAccelerometer 37  
describe, YAnButton 82  
describe, YCarbonDioxide 125  
describe, YColorLed 166

describe, YCompass 201  
describe, YCurrent 245  
describe, YDataLogger 286  
describe, YDigitalIO 361  
describe, YDisplay 409  
describe, YDualPower 490  
describe, YFiles 519  
describe, YGenericSensor 554  
describe, YGyro 604  
describe, YHubPort 657  
describe, YHumidity 688  
describe, YLed 729  
describe, YLightSensor 763  
describe, YMagnetometer 806  
describe, YModule 857  
describe, YNetwork 904  
describe, YOsControl 962  
describe, YPower 991  
describe, YPressure 1038  
describe, YPwmOutput 1080  
describe, YPwmPowerSource 1121  
describe, YQt 1150  
describe, YRealTimeClock 1191  
describe, YRefFrame 1224  
describe, YRelay 1264  
describe, YSensor 1304  
describe, YServo 1346  
describe, YTemperature 1386  
describe, YTilt 1431  
describe, YVoc 1474  
describe, YVoltage 1517  
describe, YVSource 1558  
describe, YWakeUpMonitor 1595  
describe, YWakeUpSchedule 1634  
describe, YWatchdog 1676  
describe, YWireless 1724  
DigitalIO 356  
DisableExceptions, YAPI 15  
Display 404  
DisplayLayer 455  
Données 318, 328, 341  
download, YFiles 520  
download, YModule 858  
download\_async, YFiles 521  
drawBar, YDisplayLayer 459  
drawBitmap, YDisplayLayer 460  
drawCircle, YDisplayLayer 461  
drawDisc, YDisplayLayer 462  
drawImage, YDisplayLayer 463  
drawPixel, YDisplayLayer 464  
drawRect, YDisplayLayer 465  
drawText, YDisplayLayer 466  
dutyCycleMove, YPwmOutput 1081

## E

EnableExceptions, YAPI 16  
Enregistrées 328, 341  
Erreurs 8

## F

fade, YDisplay 410  
Files 516  
FindAccelerometer, YAccelerometer 34  
FindAnButton, YAnButton 80  
FindCarbonDioxide, YCarbonDioxide 122  
FindColorLed, YColorLed 164  
FindCompass, YCompass 198  
FindCurrent, YCurrent 242  
FindDataLogger, YDataLogger 284  
FindDigitalIO, YDigitalIO 358  
FindDisplay, YDisplay 406  
FindDualPower, YDualPower 488  
FindFiles, YFiles 517  
FindGenericSensor, YGenericSensor 551  
FindGyro, YGyro 601  
FindHubPort, YHubPort 655  
FindHumidity, YHumidity 685  
FindLed, YLed 727  
FindLightSensor, YLightSensor 759  
FindMagnetometer, YMagnetometer 803  
FindModule, YModule 855  
FindNetwork, YNetwork 901  
FindOsControl, YOsControl 960  
FindPower, YPower 988  
FindPressure, YPressure 1035  
FindPwmOutput, YPwmOutput 1078  
FindPwmPowerSource, YPwmPowerSource 1119  
FindQt, YQt 1147  
FindRealTimeClock, YRealTimeClock 1189  
FindRefFrame, YRefFrame 1221  
FindRelay, YRelay 1261  
FindSensor, YSensor 1301  
FindServo,YServo 1344  
FindTemperature, YTemperature 1383  
FindTilt, YTilt 1428  
FindVoc, YVoc 1471  
FindVoltage, YVoltage 1514  
FindVSource, YVSource 1556  
FindWakeUpMonitor, YWakeUpMonitor 1593  
FindWakeUpSchedule, YWakeUpSchedule 1632  
FindWatchdog, YWatchdog 1673  
FindWireless, YWireless 1721  
FirstAccelerometer, YAccelerometer 35  
FirstAnButton, YAnButton 81  
FirstCarbonDioxide, YCarbonDioxide 123  
FirstColorLed, YColorLed 165  
FirstCompass, YCompass 199  
FirstCurrent, YCurrent 243  
FirstDataLogger, YDataLogger 285  
FirstDigitalIO, YDigitalIO 359  
FirstDisplay, YDisplay 407

FirstDualPower, YDualPower 489

FirstFiles, YFiles 518  
FirstGenericSensor, YGenericSensor 552  
FirstGyro, YGyro 602  
FirstHubPort, YHubPort 656  
FirstHumidity, YHumidity 686  
FirstLed, YLed 728

FirstLightSensor, YLightSensor 760  
FirstMagnetometer, YMagnetometer 804  
FirstModule, YModule 856  
FirstNetwork, YNetwork 902  
FirstOsControl, YOsControl 961  
FirstPower, YPower 989  
FirstPressure, YPressure 1036  
FirstPwmOutput, YPwmOutput 1079  
FirstPwmPowerSource, YPwmPowerSource 1120

FirstQt, YQt 1148  
FirstRealTimeClock, YRealTimeClock 1190

FirstRefFrame, YRefFrame 1222  
FirstRelay, YRelay 1262  
FirstSensor, YSensor 1302  
FirstServo, YServo 1345  
FirstTemperature, YTemperature 1384  
FirstTilt, YTilt 1429  
FirstVoc, YVoc 1472  
FirstVoltage, YVoltage 1515  
FirstVSource, YVSource 1557  
FirstWakeUpMonitor, YWakeUpMonitor 1594  
FirstWakeUpSchedule, YWakeUpSchedule 1633  
FirstWatchdog, YWatchdog 1674  
FirstWireless, YWireless 1722  
Fonctions 13, 1299  
forgetAllDataStreams, YDataLogger 287  
format\_fs, YFiles 522  
Forme 318  
FreeAPI, YAPI 17  
functionCount, YModule 859  
functionId, YModule 860  
functionName, YModule 861  
functionValue, YModule 862

## G

GenericSensor 549  
get\_3DCalibrationHint, YRefFrame 1225  
get\_3DCalibrationLogMsg, YRefFrame 1226  
get\_3DCalibrationProgress, YRefFrame 1227  
get\_3DCalibrationStage, YRefFrame 1228  
get\_3DCalibrationStageProgress, YRefFrame 1229  
get\_adminPassword, YNetwork 905  
get\_advertisedValue, YAccelerometer 38  
get\_advertisedValue, YAnButton 83  
get\_advertisedValue, YCarbonDioxide 126  
get\_advertisedValue, YColorLed 167  
get\_advertisedValue, YCompass 202  
get\_advertisedValue, YCurrent 246  
get\_advertisedValue, YDataLogger 288  
get\_advertisedValue, YDigitalIO 362

get\_advertisedValue, YDisplay 411  
get\_advertisedValue, YDualPower 491  
get\_advertisedValue, YFiles 523  
get\_advertisedValue, YGenericSensor 555  
get\_advertisedValue, YGyro 605  
get\_advertisedValue, YHubPort 658  
get\_advertisedValue, YHumidity 689  
get\_advertisedValue, YLed 730  
get\_advertisedValue, YLightSensor 764  
get\_advertisedValue, YMagnetometer 807  
get\_advertisedValue, YNetwork 906  
get\_advertisedValue, YOsControl 963  
get\_advertisedValue, YPower 992  
get\_advertisedValue, YPressure 1039  
get\_advertisedValue, YPwmOutput 1082  
get\_advertisedValue, YPwmPowerSource 1122  
get\_advertisedValue, YQt 1151  
get\_advertisedValue, YRealTimeClock 1192  
get\_advertisedValue, YRefFrame 1230  
get\_advertisedValue, YRelay 1265  
get\_advertisedValue, YSensor 1305  
get\_advertisedValue,YServo 1347  
get\_advertisedValue, YTemperature 1387  
get\_advertisedValue, YTilt 1432  
get\_advertisedValue, YVoc 1475  
get\_advertisedValue, YVoltage 1518  
get\_advertisedValue, YVSource 1559  
get\_advertisedValue, YWakeUpMonitor 1596  
get\_advertisedValue, YWakeUpSchedule 1635  
get\_advertisedValue, YWatchdog 1677  
get\_advertisedValue, YWireless 1725  
get\_analogCalibration, YAnButton 84  
get\_autoStart, YDataLogger 289  
get\_autoStart, YWatchdog 1678  
get\_averageValue, YDataRun 318  
get\_averageValue, YDataStream 342  
get\_averageValue, YMeasure 847  
get\_baudRate, YHubPort 659  
get\_beacon, YModule 863  
get\_bearing, YRefFrame 1231  
get\_bitDirection, YDigitalIO 363  
get\_bitOpenDrain, YDigitalIO 364  
get\_bitPolarity, YDigitalIO 365  
get\_bitState, YDigitalIO 366  
get\_blinking, YLed 731  
get\_brightness, YDisplay 412  
get\_calibratedValue, YAnButton 85  
get\_calibrationMax, YAnButton 86  
get\_calibrationMin, YAnButton 87  
get\_callbackCredentials, YNetwork 907  
get\_callbackEncoding, YNetwork 908  
get\_callbackMaxDelay, YNetwork 909  
get\_callbackMethod, YNetwork 910  
get\_callbackMinDelay, YNetwork 911  
get\_callbackUrl, YNetwork 912  
get\_channel, YWireless 1726  
get\_columnCount, YDataStream 343  
get\_columnNames, YDataStream 344  
get\_cosPhi, YPower 993  
get\_countdown, YRelay 1266  
get\_countdown, YWatchdog 1679  
get\_currentRawValue, YAccelerometer 39  
get\_currentRawValue, YCarbonDioxide 127  
get\_currentRawValue, YCompass 203  
get\_currentRawValue, YCurrent 247  
get\_currentRawValue, YGenericSensor 556  
get\_currentRawValue, YGyro 606  
get\_currentRawValue, YHumidity 690  
get\_currentRawValue, YLightSensor 765  
get\_currentRawValue, YMagnetometer 808  
get\_currentRawValue, YPower 994  
get\_currentRawValue, YPressure 1040  
get\_currentRawValue, YQt 1152  
get\_currentRawValue, YSensor 1306  
get\_currentRawValue, YTemperature 1388  
get\_currentRawValue, YTilt 1433  
get\_currentRawValue, YVoc 1476  
get\_currentRawValue, YVoltage 1519  
get\_currentRunIndex, YDataLogger 290  
get\_currentValue, YAccelerometer 40  
get\_currentValue, YCarbonDioxide 128  
get\_currentValue, YCompass 204  
get\_currentValue, YCurrent 248  
get\_currentValue, YGenericSensor 557  
get\_currentValue, YGyro 607  
get\_currentValue, YHumidity 691  
get\_currentValue, YLightSensor 766  
get\_currentValue, YMagnetometer 809  
get\_currentValue, YPower 995  
get\_currentValue, YPressure 1041  
get\_currentValue, YQt 1153  
get\_currentValue, YSensor 1307  
get\_currentValue, YTemperature 1389  
get\_currentValue, YTilt 1434  
get\_currentValue, YVoc 1477  
get\_currentValue, YVoltage 1520  
get\_data, YDataStream 345  
get\_dataRows, YDataStream 346  
get\_dataSamplesIntervalMs, YDataStream 347  
get\_dataSets, YDataLogger 291  
get\_dataStreams, YDataLogger 292  
get\_dateTime, YRealTimeClock 1193  
get\_detectedWlans, YWireless 1727  
get\_discoverable, YNetwork 913  
get\_display, YDisplayLayer 467  
get\_displayHeight, YDisplay 413  
get\_displayHeight, YDisplayLayer 468  
get\_displayLayer, YDisplay 414  
get\_displayType, YDisplay 415  
get\_displayWidth, YDisplay 416  
get\_displayWidth, YDisplayLayer 469  
get\_duration, YDataRun 319  
get\_duration, YDataStream 348  
get\_dutyCycle, YPwmOutput 1083  
get\_dutyCycleAtPowerOn, YPwmOutput 1084  
get\_enabled, YDisplay 417  
get\_enabled, YHubPort 660  
get\_enabled, YPwmOutput 1085

get\_enabled, YServo 1348  
get\_enabledAtPowerOn, YPwmOutput 1086  
get\_enabledAtPowerOn, YServo 1349  
get\_endTimeUTC, YDataSet 329  
get\_endTimeUTC, YMeasure 848  
get\_errorMessage, YAccelerometer 41  
get\_errorMessage, YAnButton 88  
get\_errorMessage, YCarbonDioxide 129  
get\_errorMessage, YColorLed 168  
get\_errorMessage, YCompass 205  
get\_errorMessage, YCurrent 249  
get\_errorMessage, YDataLogger 293  
get\_errorMessage, YDigitalIO 367  
get\_errorMessage, YDisplay 418  
get\_errorMessage, YDualPower 492  
get\_errorMessage, YFiles 524  
get\_errorMessage, YGenericSensor 558  
get\_errorMessage, YGyro 608  
get\_errorMessage, YHubPort 661  
get\_errorMessage, YHumidity 692  
get\_errorMessage, YLed 732  
get\_errorMessage, YLightSensor 767  
get\_errorMessage, YMagnetometer 810  
get\_errorMessage, YModule 864  
get\_errorMessage, YNetwork 914  
get\_errorMessage, YOsControl 964  
get\_errorMessage, YPower 996  
get\_errorMessage, YPressure 1042  
get\_errorMessage, YPwmOutput 1087  
get\_errorMessage, YPwmPowerSource 1123  
get\_errorMessage, YQt 1154  
get\_errorMessage, YRealTimeClock 1194  
get\_errorMessage, YRefFrame 1232  
get\_errorMessage, YRelay 1267  
get\_errorMessage, YSensor 1308  
get\_errorMessage, YServo 1350  
get\_errorMessage, YTemperature 1390  
get\_errorMessage, YTilt 1435  
get\_errorMessage, YVoc 1478  
get\_errorMessage, YVoltage 1521  
get\_errorMessage, YVSource 1560  
get\_errorMessage, YWakeUpMonitor 1597  
get\_errorMessage, YWakeUpSchedule 1636  
get\_errorMessage, YWatchdog 1680  
get\_errorMessage, YWireless 1728  
get\_errorType, YAccelerometer 42  
get\_errorType, YAnButton 89  
get\_errorType, YCarbonDioxide 130  
get\_errorType, YColorLed 169  
get\_errorType, YCompass 206  
get\_errorType, YCurrent 250  
get\_errorType, YDataLogger 294  
get\_errorType, YDigitalIO 368  
get\_errorType, YDisplay 419  
get\_errorType, YDualPower 493  
get\_errorType, YFiles 525  
get\_errorType, YGenericSensor 559  
get\_errorType, YGyro 609  
get\_errorType, YHubPort 662  
get\_errorType, YHumidity 693  
get\_errorType, YLed 733  
get\_errorType, YLightSensor 768  
get\_errorType, YMagnetometer 811  
get\_errorType, YModule 865  
get\_errorType, YNetwork 915  
get\_errorType, YOsControl 965  
get\_errorType, YPower 997  
get\_errorType, YPressure 1043  
get\_errorType, YPwmOutput 1088  
get\_errorType, YPwmPowerSource 1124  
get\_errorType, YQt 1155  
get\_errorType, YRealTimeClock 1195  
get\_errorType, YRefFrame 1233  
get\_errorType, YRelay 1268  
get\_errorType, YSensor 1309  
get\_errorType, YServo 1351  
get\_errorType, YTemperature 1391  
get\_errorType, YTilt 1436  
get\_errorType, YVoc 1479  
get\_errorType, YVoltage 1522  
get\_errorType, YVSource 1561  
get\_errorType, YWakeUpMonitor 1598  
get\_errorType, YWakeUpSchedule 1637  
get\_errorType, YWatchdog 1681  
get\_errorType, YWireless 1729  
get\_extPowerFailure, YVSource 1562  
get\_extVoltage, YDualPower 494  
get\_failure, YVSource 1563  
get\_filesCount, YFiles 526  
get\_firmwareRelease, YModule 866  
get\_freeSpace, YFiles 527  
get\_frequency, YPwmOutput 1089  
get\_friendlyName, YAccelerometer 43  
get\_friendlyName, YAnButton 90  
get\_friendlyName, YCarbonDioxide 131  
get\_friendlyName, YColorLed 170  
get\_friendlyName, YCompass 207  
get\_friendlyName, YCurrent 251  
get\_friendlyName, YDataLogger 295  
get\_friendlyName, YDigitalIO 369  
get\_friendlyName, YDisplay 420  
get\_friendlyName, YDualPower 495  
get\_friendlyName, YFiles 528  
get\_friendlyName, YGenericSensor 560  
get\_friendlyName, YGyro 610  
get\_friendlyName, YHubPort 663  
get\_friendlyName, YHumidity 694  
get\_friendlyName, YLed 734  
get\_friendlyName, YLightSensor 769  
get\_friendlyName, YMagnetometer 812  
get\_friendlyName, YNetwork 916  
get\_friendlyName, YOsControl 966  
get\_friendlyName, YPower 998  
get\_friendlyName, YPressure 1044  
get\_friendlyName, YPwmOutput 1090  
get\_friendlyName, YPwmPowerSource 1125  
get\_friendlyName, YQt 1156  
get\_friendlyName, YRealTimeClock 1196

get\_friendlyName, YRefFrame 1234  
get\_friendlyName, YRelay 1269  
get\_friendlyName,YSensor 1310  
get\_friendlyName,YServo 1352  
get\_friendlyName,YTemperature 1392  
get\_friendlyName,YTilt 1437  
get\_friendlyName,YVoc 1480  
get\_friendlyName,YVoltage 1523  
get\_friendlyName,YVSource 1564  
get\_friendlyName,YWakeUpMonitor 1599  
get\_friendlyName,YWakeUpSchedule 1638  
get\_friendlyName,YWatchdog 1682  
get\_friendlyName,YWireless 1730  
get\_functionDescriptor, YAccelerometer 44  
get\_functionDescriptor, YAnButton 91  
get\_functionDescriptor, YCarbonDioxide 132  
get\_functionDescriptor, YColorLed 171  
get\_functionDescriptor, YCompass 208  
get\_functionDescriptor, YCurrent 252  
get\_functionDescriptor, YDataLogger 296  
get\_functionDescriptor, YDigitalIO 370  
get\_functionDescriptor, YDisplay 421  
get\_functionDescriptor, YDualPower 496  
get\_functionDescriptor, YFiles 529  
get\_functionDescriptor, YGenericSensor 561  
get\_functionDescriptor, YGyro 611  
get\_functionDescriptor, YHubPort 664  
get\_functionDescriptor, YHumidity 695  
get\_functionDescriptor, YLed 735  
get\_functionDescriptor, YLightSensor 770  
get\_functionDescriptor, YMagnetometer 813  
get\_functionDescriptor, YNetwork 917  
get\_functionDescriptor, YOsControl 967  
get\_functionDescriptor, YPower 999  
get\_functionDescriptor, YPressure 1045  
get\_functionDescriptor, YPwmOutput 1091  
get\_functionDescriptor, YPwmPowerSource 1126  
get\_functionDescriptor, YQt 1157  
get\_functionDescriptor, YRealTimeClock 1197  
get\_functionDescriptor, YRefFrame 1235  
get\_functionDescriptor, YRelay 1270  
get\_functionDescriptor, YSensor 1311  
get\_functionDescriptor, YServo 1353  
get\_functionDescriptor, YTemperature 1393  
get\_functionDescriptor, YTilt 1438  
get\_functionDescriptor, YVoc 1481  
get\_functionDescriptor, YVoltage 1524  
get\_functionDescriptor, YVSource 1565  
get\_functionDescriptor, YWakeUpMonitor 1600  
get\_functionDescriptor, YWakeUpSchedule 1639  
get\_functionDescriptor, YWatchdog 1683  
get\_functionDescriptor, YWireless 1731  
get\_functionId, YAccelerometer 45  
get\_functionId, YAnButton 92  
get\_functionId, YCarbonDioxide 133  
get\_functionId, YColorLed 172  
get\_functionId, YCompass 209  
get\_functionId, YCurrent 253  
get\_functionId, YDataLogger 297  
get\_functionId, YDataSet 330  
get\_functionId, YDigitalIO 371  
get\_functionId, YDisplay 422  
get\_functionId, YDualPower 497  
get\_functionId, YFiles 530  
get\_functionId, YGenericSensor 562  
get\_functionId, YGyro 612  
get\_functionId, YHubPort 665  
get\_functionId, YHumidity 696  
get\_functionId, YLed 736  
get\_functionId, YLightSensor 771  
get\_functionId, YMagnetometer 814  
get\_functionId, YNetwork 918  
get\_functionId, YOsControl 968  
get\_functionId, YPower 1000  
get\_functionId, YPressure 1046  
get\_functionId, YPwmOutput 1092  
get\_functionId, YPwmPowerSource 1127  
get\_functionId, YQt 1158  
get\_functionId, YRealTimeClock 1198  
get\_functionId, YRefFrame 1236  
get\_functionId, YRelay 1271  
get\_functionId, YSensor 1312  
get\_functionId, YServo 1354  
get\_functionId, YTemperature 1394  
get\_functionId, YTilt 1439  
get\_functionId, YVoc 1482  
get\_functionId, YVoltage 1525  
get\_functionId, YVSource 1566  
get\_functionId, YWakeUpMonitor 1601  
get\_functionId, YWakeUpSchedule 1640  
get\_functionId, YWatchdog 1684  
get\_functionId, YWireless 1732  
get\_hardwareId, YAccelerometer 46  
get\_hardwareId, YAnButton 93  
get\_hardwareId, YCarbonDioxide 134  
get\_hardwareId, YColorLed 173  
get\_hardwareId, YCompass 210  
get\_hardwareId, YCurrent 254  
get\_hardwareId, YDataLogger 298  
get\_hardwareId, YDataSet 331  
get\_hardwareId, YDigitalIO 372  
get\_hardwareId, YDisplay 423  
get\_hardwareId, YDualPower 498  
get\_hardwareId, YFiles 531  
get\_hardwareId, YGenericSensor 563  
get\_hardwareId, YGyro 613  
get\_hardwareId, YHubPort 666  
get\_hardwareId, YHumidity 697  
get\_hardwareId, YLed 737  
get\_hardwareId, YLightSensor 772  
get\_hardwareId, YMagnetometer 815  
get\_hardwareId, YModule 867  
get\_hardwareId, YNetwork 919  
get\_hardwareId, YOsControl 969  
get\_hardwareId, YPower 1001  
get\_hardwareId, YPressure 1047  
get\_hardwareId, YPwmOutput 1093  
get\_hardwareId, YPwmPowerSource 1128

get\_hardwareId, YQt 1159  
get\_hardwareId, YRealTimeClock 1199  
get\_hardwareId, YRefFrame 1237  
get\_hardwareId, YRelay 1272  
get\_hardwareId,YSensor 1313  
get\_hardwareId,YServo 1355  
get\_hardwareId,YTemperature 1395  
get\_hardwareId,YTilt 1440  
get\_hardwareId,YVoc 1483  
get\_hardwareId,YVoltage 1526  
get\_hardwareId,YVSource 1567  
get\_hardwareId,YWakeUpMonitor 1602  
get\_hardwareId,YWakeUpSchedule 1641  
get\_hardwareId,YWatchdog 1685  
get\_hardwareId,YWireless 1733  
get\_heading, YGyro 614  
get\_highestValue, YAccelerometer 47  
get\_highestValue, YCarbonDioxide 135  
get\_highestValue, YCompass 211  
get\_highestValue, YCurrent 255  
get\_highestValue, YGenericSensor 564  
get\_highestValue, YGyro 615  
get\_highestValue, YHumidity 698  
get\_highestValue, YLightSensor 773  
get\_highestValue, YMagnetometer 816  
get\_highestValue, YPower 1002  
get\_highestValue, YPressure 1048  
get\_highestValue, YQt 1160  
get\_highestValue,YSensor 1314  
get\_highestValue,YTemperature 1396  
get\_highestValue,YTilt 1441  
get\_highestValue,YVoc 1484  
get\_highestValue,YVoltage 1527  
get\_hours, YWakeUpSchedule 1642  
get\_hslColor, YColorLed 174  
get\_icon2d, YModule 868  
get\_ipAddress, YNetwork 920  
get\_isPressed, YAnButton 94  
get\_lastLogs, YModule 869  
get\_lastTimePressed, YAnButton 95  
get\_lastTimeReleased, YAnButton 96  
get\_layerCount, YDisplay 424  
get\_layerHeight, YDisplay 425  
get\_layerHeight, YDisplayLayer 470  
get\_layerWidth, YDisplay 426  
get\_layerWidth, YDisplayLayer 471  
get\_linkQuality, YWireless 1734  
get\_list, YFiles 532  
get\_logFrequency, YAccelerometer 48  
get\_logFrequency, YCarbonDioxide 136  
get\_logFrequency, YCompass 212  
get\_logFrequency, YCurrent 256  
get\_logFrequency, YGenericSensor 565  
get\_logFrequency, YGyro 616  
get\_logFrequency, YHumidity 699  
get\_logFrequency, YLightSensor 774  
get\_logFrequency, YMagnetometer 817  
get\_logFrequency, YPower 1003  
get\_logFrequency, YPressure 1049  
get\_logFrequency, YQt 1161  
get\_logFrequency, YSensor 1315  
get\_logFrequency, YTemperature 1397  
get\_logFrequency, YTilt 1442  
get\_logFrequency, YVoc 1485  
get\_logFrequency, YVoltage 1528  
get\_logicalName, YAccelerometer 49  
get\_logicalName, YAnButton 97  
get\_logicalName, YCarbonDioxide 137  
get\_logicalName, YColorLed 175  
get\_logicalName, YCompass 213  
get\_logicalName, YCurrent 257  
get\_logicalName, YDataLogger 299  
get\_logicalName, YDigitalIO 373  
get\_logicalName, YDisplay 427  
get\_logicalName, YDualPower 499  
get\_logicalName, YFiles 533  
get\_logicalName, YGenericSensor 566  
get\_logicalName, YGyro 617  
get\_logicalName, YHubPort 667  
get\_logicalName, YHumidity 700  
get\_logicalName, YLed 738  
get\_logicalName, YLightSensor 775  
get\_logicalName, YMagnetometer 818  
get\_logicalName, YModule 870  
get\_logicalName, YNetwork 921  
get\_logicalName, YOsControl 970  
get\_logicalName, YPower 1004  
get\_logicalName, YPressure 1050  
get\_logicalName, YPwmOutput 1094  
get\_logicalName, YPwmPowerSource 1129  
get\_logicalName, YQt 1162  
get\_logicalName, YRealTimeClock 1200  
get\_logicalName, YRefFrame 1238  
get\_logicalName, YRelay 1273  
get\_logicalName, YSensor 1316  
get\_logicalName, YServo 1356  
get\_logicalName, YTemperature 1398  
get\_logicalName, YTilt 1443  
get\_logicalName, YVoc 1486  
get\_logicalName, YVoltage 1529  
get\_logicalName, YVSource 1568  
get\_logicalName, YWakeUpMonitor 1603  
get\_logicalName, YWakeUpSchedule 1643  
get\_logicalName, YWatchdog 1686  
get\_logicalName, YWireless 1735  
get\_lowestValue, YAccelerometer 50  
get\_lowestValue, YCarbonDioxide 138  
get\_lowestValue, YCompass 214  
get\_lowestValue, YCurrent 258  
get\_lowestValue, YGenericSensor 567  
get\_lowestValue, YGyro 618  
get\_lowestValue, YHumidity 701  
get\_lowestValue, YLightSensor 776  
get\_lowestValue, YMagnetometer 819  
get\_lowestValue, YPower 1005  
get\_lowestValue, YPressure 1051  
get\_lowestValue, YQt 1163  
get\_lowestValue, YSensor 1317

get\_lowestValue, YTemperature 1399  
get\_lowestValue, YTilt 1444  
get\_lowestValue, YVoc 1487  
get\_lowestValue, YVoltage 1530  
get\_luminosity, YLed 739  
get\_luminosity, YModule 871  
get\_macAddress, YNetwork 922  
get\_magneticHeading, YCompass 215  
get\_maxTimeOnStateA, YRelay 1274  
get\_maxTimeOnStateA, YWatchdog 1687  
get\_maxTimeOnStateB, YRelay 1275  
get\_maxTimeOnStateB, YWatchdog 1688  
get\_maxValue, YDataRun 320  
get\_maxValue, YDataStream 349  
get\_maxValue, YMeasure 849  
get\_measureNames, YDataRun 321  
get\_measures, YDataSet 332  
get\_message, YWireless 1736  
get\_meter, YPower 1006  
get\_meterTimer, YPower 1007  
get\_minutes, YWakeUpSchedule 1644  
get\_minutesA, YWakeUpSchedule 1645  
get\_minutesB, YWakeUpSchedule 1646  
get\_minValue, YDataRun 322  
get\_minValue, YDataStream 350  
get\_minValue, YMeasure 850  
get\_module, YAccelerometer 51  
get\_module, YAnButton 98  
get\_module, YCarbonDioxide 139  
get\_module, YColorLed 176  
get\_module, YCompass 216  
get\_module, YCurrent 259  
get\_module, YDataLogger 300  
get\_module, YDigitalIO 374  
get\_module, YDisplay 428  
get\_module, YDualPower 500  
get\_module, YFiles 534  
get\_module, YGenericSensor 568  
get\_module, YGyro 619  
get\_module, YHubPort 668  
get\_module, YHumidity 702  
get\_module, YLed 740  
get\_module, YLightSensor 777  
get\_module, YMagnetometer 820  
get\_module, YNetwork 923  
get\_module, YOsControl 971  
get\_module, YPower 1008  
get\_module, YPressure 1052  
get\_module, YPwmOutput 1095  
get\_module, YPwmPowerSource 1130  
get\_module, YQt 1164  
get\_module, YRealTimeClock 1201  
get\_module, YRefFrame 1239  
get\_module, YRelay 1276  
get\_module,YSensor 1318  
get\_module, YServo 1357  
get\_module, YTemperature 1400  
get\_module, YTilt 1445  
get\_module, YVoc 1488  
get\_module, YVoltage 1531  
get\_module, YVSource 1569  
get\_module, YWakeUpMonitor 1604  
get\_module, YWakeUpSchedule 1647  
get\_module, YWatchdog 1689  
get\_module, YWireless 1737  
get\_module\_async, YAccelerometer 52  
get\_module\_async, YAnButton 99  
get\_module\_async, YCarbonDioxide 140  
get\_module\_async, YColorLed 177  
get\_module\_async, YCompass 217  
get\_module\_async, YCurrent 260  
get\_module\_async, YDataLogger 301  
get\_module\_async, YDigitalIO 375  
get\_module\_async, YDisplay 429  
get\_module\_async, YDualPower 501  
get\_module\_async, YFiles 535  
get\_module\_async, YGenericSensor 569  
get\_module\_async, YGyro 620  
get\_module\_async, YHubPort 669  
get\_module\_async, YHumidity 703  
get\_module\_async, YLed 741  
get\_module\_async, YLightSensor 778  
get\_module\_async, YMagnetometer 821  
get\_module\_async, YNetwork 924  
get\_module\_async, YOsControl 972  
get\_module\_async, YPower 1009  
get\_module\_async, YPressure 1053  
get\_module\_async, YPwmOutput 1096  
get\_module\_async, YPwmPowerSource 1131  
get\_module\_async, YQt 1165  
get\_module\_async, YRealTimeClock 1202  
get\_module\_async, YRefFrame 1240  
get\_module\_async, YRelay 1277  
get\_module\_async, YSensor 1319  
get\_module\_async, YServo 1358  
get\_module\_async, YTemperature 1401  
get\_module\_async, YTilt 1446  
get\_module\_async, YVoc 1489  
get\_module\_async, YVoltage 1532  
get\_module\_async, YVSource 1570  
get\_module\_async, YWakeUpMonitor 1605  
get\_module\_async, YWakeUpSchedule 1648  
get\_module\_async, YWatchdog 1690  
get\_module\_async, YWireless 1738  
get\_monthDays, YWakeUpSchedule 1649  
get\_months, YWakeUpSchedule 1650  
get\_mountOrientation, YRefFrame 1241  
get\_mountPosition, YRefFrame 1242  
get\_neutral, YServo 1359  
get\_nextOccurrence, YWakeUpSchedule 1651  
get\_nextWakeUp, YWakeUpMonitor 1606  
get\_orientation, YDisplay 430  
get\_output, YRelay 1278  
get\_output, YWatchdog 1691  
get\_outputVoltage, YDigitalIO 376  
get\_overCurrent, YVSource 1571  
get\_overHeat, YVSource 1572  
get\_overLoad, YVSource 1573

get\_period, YPwmOutput 1097  
get\_persistentSettings, YModule 872  
get\_pitch, YGyro 621  
get\_poeCurrent, YNetwork 925  
get\_portDirection, YDigitalIO 377  
get\_portOpenDrain, YDigitalIO 378  
get\_portPolarity, YDigitalIO 379  
get\_portSize, YDigitalIO 380  
get\_portState, YDigitalIO 381  
get\_portState, YHubPort 670  
get\_position, YServo 1360  
get\_positionAtPowerOn, YServo 1361  
get\_power, YLed 742  
get\_powerControl, YDualPower 502  
get\_powerDuration, YWakeUpMonitor 1607  
get\_powerMode, YPwmPowerSource 1132  
get\_powerState, YDualPower 503  
get\_preview, YDataSet 333  
get\_primaryDNS, YNetwork 926  
get\_productId, YModule 873  
get\_productName, YModule 874  
get\_productRelease, YModule 875  
get\_progress, YDataSet 334  
get\_pulseCounter, YAnButton 100  
get\_pulseDuration, YPwmOutput 1098  
get\_pulseTimer, YAnButton 101  
get\_pulseTimer, YRelay 1279  
get\_pulseTimer, YWatchdog 1692  
get\_quaternionW, YGyro 622  
get\_quaternionX, YGyro 623  
get\_quaternionY, YGyro 624  
get\_quaternionZ, YGyro 625  
get\_range, YServo 1362  
get\_rawValue, YAnButton 102  
get\_readiness, YNetwork 927  
get\_rebootCountdown, YModule 876  
get\_recordedData, YAccelerometer 53  
get\_recordedData, YCarbonDioxide 141  
get\_recordedData, YCompass 218  
get\_recordedData, YCurrent 261  
get\_recordedData, YGenericSensor 570  
get\_recordedData, YGyro 626  
get\_recordedData, YHumidity 704  
get\_recordedData, YLightSensor 779  
get\_recordedData, YMagnetometer 822  
get\_recordedData, YPower 1010  
get\_recordedData, YPressure 1054  
get\_recordedData, YQt 1166  
get\_recordedData, YSensor 1320  
get\_recordedData, YTemperature 1402  
get\_recordedData, YTilt 1447  
get\_recordedData, YVoc 1490  
get\_recordedData, YVoltage 1533  
get\_recording, YDataLogger 302  
get\_regulationFailure, YVSource 1574  
get\_reportFrequency, YAccelerometer 54  
get\_reportFrequency, YCarbonDioxide 142  
get\_reportFrequency, YCompass 219  
get\_reportFrequency, YCurrent 262  
get\_reportFrequency, YGenericSensor 571  
get\_reportFrequency, YGyro 627  
get\_reportFrequency, YHumidity 705  
get\_reportFrequency, YLightSensor 780  
get\_reportFrequency, YMagnetometer 823  
get\_reportFrequency, YPower 1011  
get\_reportFrequency, YPressure 1055  
get\_reportFrequency, YQt 1167  
get\_reportFrequency, YSensor 1321  
get\_reportFrequency, YTemperature 1403  
get\_reportFrequency, YTilt 1448  
get\_reportFrequency, YVoc 1491  
get\_reportFrequency, YVoltage 1534  
get\_resolution, YAccelerometer 55  
get\_resolution, YCarbonDioxide 143  
get\_resolution, YCompass 220  
get\_resolution, YCurrent 263  
get\_resolution, YGenericSensor 572  
get\_resolution, YGyro 628  
get\_resolution, YHumidity 706  
get\_resolution, YLightSensor 781  
get\_resolution, YMagnetometer 824  
get\_resolution, YPower 1012  
get\_resolution, YPressure 1056  
get\_resolution, YQt 1168  
get\_resolution, YSensor 1322  
get\_resolution, YTemperature 1404  
get\_resolution, YTilt 1449  
get\_resolution, YVoc 1492  
get\_resolution, YVoltage 1535  
get\_rgbColor, YColorLed 178  
get\_rgbColorAtPowerOn, YColorLed 179  
get\_roll, YGyro 629  
get\_router, YNetwork 928  
getRowCount, YDataStream 351  
get\_runIndex, YDataStream 352  
get\_running, YWatchdog 1693  
get\_secondaryDNS, YNetwork 929  
get\_security, YWireless 1739  
get\_sensitivity, YAnButton 103  
get\_sensorType, YTemperature 1405  
get\_serialNumber, YModule 877  
get\_shutdownCountdown, YOsControl 973  
get\_signalRange, YGenericSensor 573  
get\_signalUnit, YGenericSensor 574  
get\_signalValue, YGenericSensor 575  
get\_sleepCountdown, YWakeUpMonitor 1608  
get\_ssId, YWireless 1740  
get\_startTime, YDataStream 353  
get\_startTimeUTC, YDataRun 323  
get\_startTimeUTC, YDataSet 335  
get\_startTimeUTC, YDataStream 354  
get\_startTimeUTC, YMeasure 851  
get\_startupSeq, YDisplay 431  
get\_state, YRelay 1280  
get\_state, YWatchdog 1694  
get\_stateAtPowerOn, YRelay 1281  
get\_stateAtPowerOn, YWatchdog 1695  
get\_subnetMask, YNetwork 930

get\_summary, YDataSet 336  
get\_timeSet, YRealTimeClock 1203  
get\_timeUTC, YDataLogger 303  
get\_triggerDelay, YWatchdog 1696  
get\_triggerDuration, YWatchdog 1697  
get\_unit, YAccelerometer 56  
get\_unit, YCarbonDioxide 144  
get\_unit, YCompass 221  
get\_unit, YCurrent 264  
get\_unit, YDataSet 337  
get\_unit, YGenericSensor 576  
get\_unit, YGyro 630  
get\_unit, YHumidity 707  
get\_unit, YLightSensor 782  
get\_unit, YMagnetometer 825  
get\_unit, YPower 1013  
get\_unit, YPressure 1057  
get\_unit, YQt 1169  
get\_unit,YSensor 1323  
get\_unit, YTemperature 1406  
get\_unit, YTilt 1450  
get\_unit, YVoc 1493  
get\_unit, YVoltage 1536  
get\_unit, YVSource 1575  
get\_unixTime, YRealTimeClock 1204  
get\_upTime, YModule 878  
get\_usbBandwidth, YModule 879  
get\_usbCurrent, YModule 880  
get\_userData, YAccelerometer 57  
get\_userData, YAnButton 104  
get\_userData, YCarbonDioxide 145  
get\_userData, YColorLed 180  
get\_userData, YCompass 222  
get\_userData, YCurrent 265  
get\_userData, YDataLogger 304  
get\_userData, YDigitalIO 382  
get\_userData, YDisplay 432  
get\_userData, YDualPower 504  
get\_userData, YFiles 536  
get\_userData, YGenericSensor 577  
get\_userData, YGyro 631  
get\_userData, YHubPort 671  
get\_userData, YHumidity 708  
get\_userData, YLed 743  
get\_userData, YLightSensor 783  
get\_userData, YMagnetometer 826  
get\_userData, YModule 881  
get\_userData, YNetwork 931  
get\_userData, YOsControl 974  
get\_userData, YPower 1014  
get\_userData, YPressure 1058  
get\_userData, YPwmOutput 1099  
get\_userData, YPwmPowerSource 1133  
get\_userData, YQt 1170  
get\_userData, YRealTimeClock 1205  
get\_userData, YRefFrame 1243  
get\_userData, YRelay 1282  
get\_userData, YSensor 1324  
get\_userData, YServo 1363

get(userData, YTemperature 1407  
get(userData, YTilt 1451  
get(userData, YVoc 1494  
get(userData, YVoltage 1537  
get(userData, YVSource 1576  
get(userData, YWakeUpMonitor 1609  
get(userData, YWakeUpSchedule 1652  
get(userData, YWatchdog 1698  
get(userData, YWireless 1741  
get(userPassword, YNetwork 932  
get\_utcOffset, YRealTimeClock 1206  
get\_valueCount, YDataRun 324  
get\_valueInterval, YDataRun 325  
get\_valueRange, YGenericSensor 578  
get\_voltage, YVSource 1577  
get\_wakeUpReason, YWakeUpMonitor 1610  
get\_wakeUpState, YWakeUpMonitor 1611  
get\_weekDays, YWakeUpSchedule 1653  
get\_wwwWatchdogDelay, YNetwork 933  
get\_xValue, YAccelerometer 58  
get\_xValue, YGyro 632  
get\_xValue, YMagnetometer 827  
get\_yValue, YAccelerometer 59  
get\_yValue, YGyro 633  
get\_yValue, YMagnetometer 828  
get\_zValue, YAccelerometer 60  
get\_zValue, YGyro 634  
get\_zValue, YMagnetometer 829  
GetAPIVersion, YAPI 18  
GetTickCount, YAPI 19  
Gyro 599

## H

HandleEvents, YAPI 20  
hide, YDisplayLayer 472  
Horloge 1188  
hslMove, YColorLed 181  
Humidity 683

## I

InitAPI, YAPI 21  
Interface 32, 78, 120, 163, 196, 240, 283, 356, 404, 455, 487, 516, 549, 599, 654, 683, 726, 757, 801, 853, 898, 986, 1033, 1076, 1118, 1145, 1188, 1259, 1299, 1342, 1381, 1426, 1469, 1512, 1555, 1591, 1630, 1671, 1720  
Introduction 1  
isOnline, YAccelerometer 61  
isOnline, YAnButton 105  
isOnline, YCarbonDioxide 146  
isOnline, YColorLed 182  
isOnline, YCompass 223  
isOnline, YCurrent 266  
isOnline, YDataLogger 305  
isOnline, YDigitalIO 383  
isOnline, YDisplay 433  
isOnline, YDualPower 505  
isOnline, YFiles 537

isOnline, YGenericSensor 579  
isOnline, YGyro 635  
isOnline, YHubPort 672  
isOnline, YHumidity 709  
isOnline, YLed 744  
isOnline, YLightSensor 784  
isOnline, YMagnetometer 830  
isOnline, YModule 882  
isOnline, YNetwork 934  
isOnline, YOsControl 975  
isOnline, YPower 1015  
isOnline, YPressure 1059  
isOnline, YPwmOutput 1100  
isOnline, YPwmPowerSource 1134  
isOnline, YQt 1171  
isOnline, YRealTimeClock 1207  
isOnline, YRefFrame 1244  
isOnline, YRelay 1283  
isOnline,YSensor 1325  
isOnline,YServo 1364  
isOnline,YTemperature 1408  
isOnline,YTilt 1452  
isOnline,YVoc 1495  
isOnline,YVoltage 1538  
isOnline,YVSource 1578  
isOnline,YWakeUpMonitor 1612  
isOnline,YWakeUpSchedule 1654  
isOnline,YWatchdog 1699  
isOnline,YWireless 1742  
isOnline\_async, YAccelerometer 62  
isOnline\_async, YAnButton 106  
isOnline\_async, YCarbonDioxide 147  
isOnline\_async, YColorLed 183  
isOnline\_async, YCompass 224  
isOnline\_async, YCurrent 267  
isOnline\_async, YDataLogger 306  
isOnline\_async, YDigitalIO 384  
isOnline\_async, YDisplay 434  
isOnline\_async, YDualPower 506  
isOnline\_async, YFiles 538  
isOnline\_async, YGenericSensor 580  
isOnline\_async, YGyro 636  
isOnline\_async, YHubPort 673  
isOnline\_async, YHumidity 710  
isOnline\_async, YLed 745  
isOnline\_async, YLightSensor 785  
isOnline\_async, YMagnetometer 831  
isOnline\_async, YModule 883  
isOnline\_async, YNetwork 935  
isOnline\_async, YOsControl 976  
isOnline\_async, YPower 1016  
isOnline\_async, YPressure 1060  
isOnline\_async, YPwmOutput 1101  
isOnline\_async, YPwmPowerSource 1135  
isOnline\_async, YQt 1172  
isOnline\_async, YRealTimeClock 1208  
isOnline\_async, YRefFrame 1245  
isOnline\_async, YRelay 1284  
isOnline\_async, YSensor 1326

isOnline\_async, YServo 1365  
isOnline\_async, YTTemperature 1409  
isOnline\_async, YTilt 1453  
isOnline\_async, YVoc 1496  
isOnline\_async, YVoltage 1539  
isOnline\_async, YVSource 1579  
isOnline\_async, YWakeUpMonitor 1613  
isOnline\_async, YWakeUpSchedule 1655  
isOnline\_async, YWatchdog 1700  
isOnline\_async, YWireless 1743

## J

Javascript 3  
joinNetwork, YWireless 1744

## L

LightSensor 757  
lineTo, YDisplayLayer 473  
load, YAccelerometer 63  
load, YAnButton 107  
load, YCarbonDioxide 148  
load, YColorLed 184  
load, YCompass 225  
load, YCurrent 268  
load, YDataLogger 307  
load, YDigitalIO 385  
load, YDisplay 435  
load, YDualPower 507  
load, YFiles 539  
load, YGenericSensor 581  
load, YGyro 637  
load, YHubPort 674  
load, YHumidity 711  
load, YLed 746  
load, YLightSensor 786  
load, YMagnetometer 832  
load, YModule 884  
load, YNetwork 936  
load, YOsControl 977  
load, YPower 1017  
load, YPressure 1061  
load, YPwmOutput 1102  
load, YPwmPowerSource 1136  
load, YQt 1173  
load, YRealTimeClock 1209  
load, YRefFrame 1246  
load, YRelay 1285  
load, YSensor 1327  
load, YServo 1366  
load, YTTemperature 1410  
load, YTilt 1454  
load, YVoc 1497  
load, YVoltage 1540  
load, YVSource 1580  
load, YWakeUpMonitor 1614  
load, YWakeUpSchedule 1656  
load, YWatchdog 1701  
load, YWireless 1745

load\_async, YAccelerometer 65  
load\_async, YAnButton 108  
load\_async, YCarbonDioxide 150  
load\_async, YColorLed 185  
load\_async, YCompass 227  
load\_async, YCurrent 270  
load\_async, YDataLogger 308  
load\_async, YDigitalIO 386  
load\_async, YDisplay 436  
load\_async, YDualPower 508  
load\_async, YFiles 540  
load\_async, YGenericSensor 583  
load\_async, YGyro 639  
load\_async, YHubPort 675  
load\_async, YHumidity 713  
load\_async, YLed 747  
load\_async, YLightSensor 788  
load\_async, YMagnetometer 834  
load\_async, YModule 885  
load\_async, YNetwork 937  
load\_async, YOsControl 978  
load\_async, YPower 1019  
load\_async, YPressure 1063  
load\_async, YPwmOutput 1103  
load\_async, YPwmPowerSource 1137  
load\_async, YQt 1175  
load\_async, YRealTimeClock 1210  
load\_async, YRefFrame 1247  
load\_async, YRelay 1286  
load\_async, YSensor 1329  
load\_async,YServo 1367  
load\_async, YTemperature 1412  
load\_async, YTilt 1456  
load\_async, YVoc 1499  
load\_async, YVoltage 1542  
load\_async, YVSource 1581  
load\_async, YWakeUpMonitor 1615  
load\_async, YWakeUpSchedule 1657  
load\_async, YWatchdog 1702  
load\_async, YWireless 1746  
loadCalibrationPoints, YAccelerometer 64  
loadCalibrationPoints, YCarbonDioxide 149  
loadCalibrationPoints, YCompass 226  
loadCalibrationPoints, YCurrent 269  
loadCalibrationPoints, YGenericSensor 582  
loadCalibrationPoints, YGyro 638  
loadCalibrationPoints, YHumidity 712  
loadCalibrationPoints, YLightSensor 787  
loadCalibrationPoints, YMagnetometer 833  
loadCalibrationPoints, YPower 1018  
loadCalibrationPoints, YPressure 1062  
loadCalibrationPoints, YQt 1174  
loadCalibrationPoints, YSensor 1328  
loadCalibrationPoints, YTemperature 1411  
loadCalibrationPoints, YTilt 1455  
loadCalibrationPoints, YVoc 1498  
loadCalibrationPoints, YVoltage 1541  
loadMore, YDataSet 338  
loadMore\_async, YDataSet 339

## M

Magnetometer 801  
Mesurée 847  
Mise 318  
Module 5, 853  
more3DCalibration, YRefFrame 1248  
move, YServo 1368  
moveTo, YDisplayLayer 474

## N

Network 898  
newSequence, YDisplay 437  
nextAccelerometer, YAccelerometer 66  
nextAnButton, YAnButton 109  
nextCarbonDioxide, YCarbonDioxide 151  
nextColorLed, YColorLed 186  
nextCompass, YCompass 228  
nextCurrent, YCurrent 271  
nextDataLogger, YDataLogger 309  
nextDigitalIO, YDigitalIO 387  
nextDisplay, YDisplay 438  
nextDualPower, YDualPower 509  
nextFiles, YFiles 541  
nextGenericSensor, YGenericSensor 584  
nextGyro, YGyro 640  
nextHubPort, YHubPort 676  
nextHumidity, YHumidity 714  
nextLed, YLed 748  
nextLightSensor, YLightSensor 789  
nextMagnetometer, YMagnetometer 835  
nextModule, YModule 886  
nextNetwork, YNetwork 938  
nextOsControl, YOsControl 979  
nextPower, YPower 1020  
nextPressure, YPressure 1064  
nextPwmOutput, YPwmOutput 1104  
nextPwmPowerSource, YPwmPowerSource 1138  
nextQt, YQt 1176  
nextRealTimeClock, YRealTimeClock 1211  
nextRefFrame, YRefFrame 1249  
nextRelay, YRelay 1287  
nextSensor, YSensor 1330  
nextServo, YServo 1369  
nextTemperature, YTemperature 1413  
nextTilt, YTilt 1457  
nextVoc, YVoc 1500  
nextVoltage, YVoltage 1543  
nextVSource, YVSource 1582  
nextWakeUpMonitor, YWakeUpMonitor 1616  
nextWakeUpSchedule, YWakeUpSchedule 1658  
nextWatchdog, YWatchdog 1703  
nextWireless, YWireless 1747

## O

Objets 455

## P

pauseSequence, YDisplay 439  
ping, YNetwork 939  
playSequence, YDisplay 440  
Port 654  
Power 986  
Préparation 3  
PreregisterHub, YAPI 22  
Pressure 1033  
pulse, YDigitalIO 388  
pulse, YRelay 1288  
pulse, YVSource 1583  
pulse, YWatchdog 1704  
pulseDurationMove, YPwmOutput 1105  
PwmPowerSource 1118

## Q

Quaternion 1145

## R

Real 1188  
reboot, YModule 887  
Reference 12  
Référentiel 1219  
registerAnglesCallback, YGyro 641  
RegisterDeviceArrivalCallback, YAPI 23  
RegisterDeviceRemovalCallback, YAPI 24  
RegisterHub, YAPI 25  
registerQuaternionCallback, YGyro 642  
registerTimedReportCallback, YAccelerometer 67  
registerTimedReportCallback, YCarbonDioxide 152  
registerTimedReportCallback, YCompass 229  
registerTimedReportCallback, YCurrent 272  
registerTimedReportCallback, YGenericSensor 585  
registerTimedReportCallback, YGyro 643  
registerTimedReportCallback, YHumidity 715  
registerTimedReportCallback, YLightSensor 790  
registerTimedReportCallback, YMagnetometer 836  
registerTimedReportCallback, YPower 1021  
registerTimedReportCallback, YPressure 1065  
registerTimedReportCallback, YQt 1177  
registerTimedReportCallback,YSensor 1331  
registerTimedReportCallback, YTilt 1414  
registerTimedReportCallback, YTilt 1458  
registerTimedReportCallback, YVoc 1501  
registerTimedReportCallback, YVoltage 1544  
registerValueCallback, YAccelerometer 68  
registerValueCallback, YAnButton 110  
registerValueCallback, YCarbonDioxide 153  
registerValueCallback, YColorLed 187  
registerValueCallback, YCompass 230  
registerValueCallback, YCurrent 273

registerValueCallback, YDataLogger 310  
registerValueCallback, YDigitalIO 389  
registerValueCallback, YDisplay 441  
registerValueCallback, YDualPower 510  
registerValueCallback, YFiles 542  
registerValueCallback, YGenericSensor 586  
registerValueCallback, YGyro 644  
registerValueCallback, YHubPort 677  
registerValueCallback, YHumidity 716  
registerValueCallback, YLed 749  
registerValueCallback, YLightSensor 791  
registerValueCallback, YMagnetometer 837  
registerValueCallback, YNetwork 940  
registerValueCallback, YOsControl 980  
registerValueCallback, YPower 1022  
registerValueCallback, YPressure 1066  
registerValueCallback, YPwmOutput 1106  
registerValueCallback, YPwmPowerSource 1139  
registerValueCallback, YQt 1178  
registerValueCallback, YRealTimeClock 1212  
registerValueCallback, YRefFrame 1250  
registerValueCallback, YRelay 1289  
registerValueCallback, YSensor 1332  
registerValueCallback, YServo 1370  
registerValueCallback, YTilt 1459  
registerValueCallback, YVoc 1502  
registerValueCallback, YVoltage 1545  
registerValueCallback, YVSource 1584  
registerValueCallback, YWakeUpMonitor 1617  
registerValueCallback, YWakeUpSchedule 1659  
registerValueCallback, YWatchdog 1705  
registerValueCallback, YWireless 1748  
Relay 1259  
remove, YFiles 543  
reset, YDisplayLayer 475  
reset, YPower 1023  
resetAll, YDisplay 442  
resetCounter, YAnButton 111  
resetSleepCountDown, YWakeUpMonitor 1618  
resetWatchdog, YWatchdog 1706  
revertFromFlash, YModule 888  
rgbMove, YColorLed 188

## S

save3DCalibration, YRefFrame 1251  
saveSequence, YDisplay 443  
saveToFlash, YModule 889  
selectColorPen, YDisplayLayer 476  
selectEraser, YDisplayLayer 477  
selectFont, YDisplayLayer 478  
selectGrayPen, YDisplayLayer 479  
Senseur 1299  
Séquence 318, 328, 341  
Servo 1342  
set\_adminPassword, YNetwork 941  
set\_analogCalibration, YAnButton 112  
set\_autoStart, YDataLogger 311  
set\_autoStart, YWatchdog 1707

set\_beacon, YModule 890  
set\_bearing, YRefFrame 1252  
set\_bitDirection, YDigitalIO 390  
set\_bitOpenDrain, YDigitalIO 391  
set\_bitPolarity, YDigitalIO 392  
set\_bitState, YDigitalIO 393  
set\_blinking, YLed 750  
set\_brightness, YDisplay 444  
set\_calibrationMax, YAnButton 113  
set\_calibrationMin, YAnButton 114  
set\_callbackCredentials, YNetwork 942  
set\_callbackEncoding, YNetwork 943  
set\_callbackMaxDelay, YNetwork 944  
set\_callbackMethod, YNetwork 945  
set\_callbackMinDelay, YNetwork 946  
set\_callbackUrl, YNetwork 947  
set\_discoverable, YNetwork 948  
set\_dutyCycle, YPwmOutput 1107  
set\_dutyCycleAtPowerOn, YPwmOutput 1108  
set\_enabled, YDisplay 445  
set\_enabled, YHubPort 678  
set\_enabled, YPwmOutput 1109  
set\_enabled,YServo 1371  
set\_enabledAtPowerOn, YPwmOutput 1110  
set\_enabledAtPowerOn,YServo 1372  
set\_frequency, YPwmOutput 1111  
set\_highestValue, YAccelerometer 69  
set\_highestValue, YCarbonDioxide 154  
set\_highestValue, YCompass 231  
set\_highestValue, YCurrent 274  
set\_highestValue, YGenericSensor 587  
set\_highestValue, YGyro 645  
set\_highestValue, YHumidity 717  
set\_highestValue, YLightSensor 792  
set\_highestValue, YMagnetometer 838  
set\_highestValue, YPower 1024  
set\_highestValue, YPressure 1067  
set\_highestValue, YQt 1179  
set\_highestValue, YSensor 1333  
set\_highestValue, YTemperature 1416  
set\_highestValue, YTilt 1460  
set\_highestValue, YVoc 1503  
set\_highestValue, YVoltage 1546  
set\_hours, YWakeUpSchedule 1660  
set\_hslColor, YColorLed 189  
set\_logFrequency, YAccelerometer 70  
set\_logFrequency, YCarbonDioxide 155  
set\_logFrequency, YCompass 232  
set\_logFrequency, YCurrent 275  
set\_logFrequency, YGenericSensor 588  
set\_logFrequency, YGyro 646  
set\_logFrequency, YHumidity 718  
set\_logFrequency, YLightSensor 793  
set\_logFrequency, YMagnetometer 839  
set\_logFrequency, YPower 1025  
set\_logFrequency, YPressure 1068  
set\_logFrequency, YQt 1180  
set\_logFrequency, YSensor 1334  
set\_logFrequency, YTemperature 1417  
set\_logFrequency, YTilt 1461  
set\_logFrequency, YVoc 1504  
set\_logFrequency, YVoltage 1547  
set\_logicalName, YAccelerometer 71  
set\_logicalName, YAnButton 115  
set\_logicalName, YCarbonDioxide 156  
set\_logicalName, YColorLed 190  
set\_logicalName, YCompass 233  
set\_logicalName, YCurrent 276  
set\_logicalName, YDataLogger 312  
set\_logicalName, YDigitalIO 394  
set\_logicalName, YDisplay 446  
set\_logicalName, YDualPower 511  
set\_logicalName, YFiles 544  
set\_logicalName, YGenericSensor 589  
set\_logicalName, YGyro 647  
set\_logicalName, YHubPort 679  
set\_logicalName, YHumidity 719  
set\_logicalName, YLed 751  
set\_logicalName, YLightSensor 794  
set\_logicalName, YMagnetometer 840  
set\_logicalName, YModule 891  
set\_logicalName, YNetwork 949  
set\_logicalName, YOsControl 981  
set\_logicalName, YPower 1026  
set\_logicalName, YPressure 1069  
set\_logicalName, YPwmOutput 1112  
set\_logicalName, YPwmPowerSource 1140  
set\_logicalName, YQt 1181  
set\_logicalName, YRealTimeClock 1213  
set\_logicalName, YRefFrame 1253  
set\_logicalName, YRelay 1290  
set\_logicalName, YSensor 1335  
set\_logicalName, YServo 1373  
set\_logicalName, YTemperature 1418  
set\_logicalName, YTilt 1462  
set\_logicalName, YVoc 1505  
set\_logicalName, YVoltage 1548  
set\_logicalName, YVSource 1585  
set\_logicalName, YWakeUpMonitor 1619  
set\_logicalName, YWakeUpSchedule 1661  
set\_logicalName, YWatchdog 1708  
set\_logicalName, YWireless 1749  
set\_lowestValue, YAccelerometer 72  
set\_lowestValue, YCarbonDioxide 157  
set\_lowestValue, YCompass 234  
set\_lowestValue, YCurrent 277  
set\_lowestValue, YGenericSensor 590  
set\_lowestValue, YGyro 648  
set\_lowestValue, YHumidity 720  
set\_lowestValue, YLightSensor 795  
set\_lowestValue, YMagnetometer 841  
set\_lowestValue, YPower 1027  
set\_lowestValue, YPressure 1070  
set\_lowestValue, YQt 1182  
set\_lowestValue, YSensor 1336  
set\_lowestValue, YTemperature 1419  
set\_lowestValue, YTilt 1463  
set\_lowestValue, YVoc 1506

set\_lowestValue, YVoltage 1549  
set\_luminosity, YLed 752  
set\_luminosity, YModule 892  
set\_maxTimeOnStateA, YRelay 1291  
set\_maxTimeOnStateA, YWatchdog 1709  
set\_maxTimeOnStateB, YRelay 1292  
set\_maxTimeOnStateB, YWatchdog 1710  
set\_minutes, YWakeUpSchedule 1662  
set\_minutesA, YWakeUpSchedule 1663  
set\_minutesB, YWakeUpSchedule 1664  
set\_monthDays, YWakeUpSchedule 1665  
set\_months, YWakeUpSchedule 1666  
set\_mountPosition, YRefFrame 1254  
set\_neutral,YServo 1374  
set\_nextWakeUp, YWakeUpMonitor 1620  
set\_orientation, YDisplay 447  
set\_output, YRelay 1293  
set\_output, YWatchdog 1711  
set\_outputVoltage, YDigitalIO 395  
set\_period, YPwmOutput 1113  
set\_portDirection, YDigitalIO 396  
set\_portOpenDrain, YDigitalIO 397  
set\_portPolarity, YDigitalIO 398  
set\_portState, YDigitalIO 399  
set\_position, YServo 1375  
set\_positionAtPowerOn, YServo 1376  
set\_power, YLed 753  
set\_powerControl, YDualPower 512  
set\_powerDuration, YWakeUpMonitor 1621  
set\_powerMode, YPwmPowerSource 1141  
set\_primaryDNS, YNetwork 950  
set\_pulseDuration, YPwmOutput 1114  
set\_range, YServo 1377  
set\_recording, YDataLogger 313  
set\_reportFrequency, YAccelerometer 73  
set\_reportFrequency, YCarbonDioxide 158  
set\_reportFrequency, YCompass 235  
set\_reportFrequency, YCurrent 278  
set\_reportFrequency, YGenericSensor 591  
set\_reportFrequency, YGyro 649  
set\_reportFrequency, YHumidity 721  
set\_reportFrequency, YLightSensor 796  
set\_reportFrequency, YMagnetometer 842  
set\_reportFrequency, YPower 1028  
set\_reportFrequency, YPressure 1071  
set\_reportFrequency, YQt 1183  
set\_reportFrequency, YSensor 1337  
set\_reportFrequency, YTemperature 1420  
set\_reportFrequency, YTilt 1464  
set\_reportFrequency, YVoc 1507  
set\_reportFrequency, YVoltage 1550  
set\_resolution, YAccelerometer 74  
set\_resolution, YCarbonDioxide 159  
set\_resolution, YCompass 236  
set\_resolution, YCurrent 279  
set\_resolution, YGenericSensor 592  
set\_resolution, YGyro 650  
set\_resolution, YHumidity 722  
set\_resolution, YLightSensor 797  
set\_resolution, YMagnetometer 843  
set\_resolution, YPower 1029  
set\_resolution, YPressure 1072  
set\_resolution, YQt 1184  
set\_resolution, YSensor 1338  
set\_resolution, YTemperature 1421  
set\_resolution, YTilt 1465  
set\_resolution, YVoc 1508  
set\_resolution, YVoltage 1551  
set\_rgbColor, YColorLed 191  
set\_rgbColorAtPowerOn, YColorLed 192  
set\_running, YWatchdog 1712  
set\_secondaryDNS, YNetwork 951  
set\_sensitivity, YAnButton 116  
set\_sensorType, YTemperature 1422  
set\_signalRange, YGenericSensor 593  
set\_sleepCountdown, YWakeUpMonitor 1622  
set\_startupSeq, YDisplay 448  
set\_state, YRelay 1294  
set\_state, YWatchdog 1713  
set\_stateAtPowerOn, YRelay 1295  
set\_stateAtPowerOn, YWatchdog 1714  
set\_timeUTC, YDataLogger 314  
set\_triggerDelay, YWatchdog 1715  
set\_triggerDuration, YWatchdog 1716  
set\_unit, YGenericSensor 594  
set\_unixTime, YRealTimeClock 1214  
set\_usbBandwidth, YModule 893  
set\_userData, YAccelerometer 75  
set\_userData, YAnButton 117  
set\_userData, YCarbonDioxide 160  
set\_userData, YColorLed 193  
set\_userData, YCompass 237  
set\_userData, YCurrent 280  
set\_userData, YDataLogger 315  
set\_userData, YDigitalIO 400  
set\_userData, YDisplay 449  
set\_userData, YDualPower 513  
set\_userData, YFiles 545  
set\_userData, YGenericSensor 595  
set\_userData, YGyro 651  
set\_userData, YHubPort 680  
set\_userData, YHumidity 723  
set\_userData, YLed 754  
set\_userData, YLightSensor 798  
set\_userData, YMagnetometer 844  
set\_userData, YModule 894  
set\_userData, YNetwork 952  
set\_userData, YOsControl 982  
set\_userData, YPower 1030  
set\_userData, YPressure 1073  
set\_userData, YPwmOutput 1115  
set\_userData, YPwmPowerSource 1142  
set\_userData, YQt 1185  
set\_userData, YRealTimeClock 1215  
set\_userData, YRefFrame 1255  
set\_userData, YRelay 1296  
set\_userData, YSensor 1339  
set\_userData, YServo 1378

set(userData, YTemperature 1423  
set(userData, YTilt 1466  
set(userData, YVoc 1509  
set(userData, YVoltage 1552  
set(userData, YVSource 1586  
set(userData, YWakeUpMonitor 1623  
set(userData, YWakeUpSchedule 1667  
set(userData, YWatchdog 1717  
set(userData, YWireless 1750  
set(userPassword, YNetwork 953  
set\_utcOffset, YRealTimeClock 1216  
set\_valueInterval, YDataRun 326  
set\_valueRange, YGenericSensor 596  
set\_voltage, YVSource 1587  
set\_weekDays, YWakeUpSchedule 1668  
set\_wwwWatchdogDelay, YNetwork 954  
setAntialiasingMode, YDisplayLayer 480  
setConsoleBackground, YDisplayLayer 481  
setConsoleMargins, YDisplayLayer 482  
setConsoleWordWrap, YDisplayLayer 483  
setLayerPosition, YDisplayLayer 484  
SetTimeout, YAPI 26  
shutdown, YOsControl 983  
Sleep, YAPI 27  
sleep, YWakeUpMonitor 1624  
sleepFor, YWakeUpMonitor 1625  
sleepUntil, YWakeUpMonitor 1626  
Source 1555  
start3DCalibration, YRefFrame 1256  
stopSequence, YDisplay 450  
swapLayerContent, YDisplay 451

## T

Temperature 1381  
Temps 1188  
Tension 1555  
Tilt 1426  
toggle\_bitState, YDigitalIO 401  
triggerFirmwareUpdate, YModule 895  
Type 1299

## U

unhide, YDisplayLayer 485  
UnregisterHub, YAPI 28  
UpdateDeviceList, YAPI 29  
UpdateDeviceList\_async, YAPI 30  
upload, YDisplay 452  
upload, YFiles 546  
useDHCP, YNetwork 955  
useStaticIP, YNetwork 956

## V

Valeur 847  
Voltage 1512  
voltageMove, YVSource 1588

## W

wait\_async, YAccelerometer 76  
wait\_async, YAnButton 118  
wait\_async, YCarbonDioxide 161  
wait\_async, YColorLed 194  
wait\_async, YCompass 238  
wait\_async, YCurrent 281  
wait\_async, YDataLogger 316  
wait\_async, YDigitalIO 402  
wait\_async, YDisplay 453  
wait\_async, YDualPower 514  
wait\_async, YFiles 547  
wait\_async, YGenericSensor 597  
wait\_async, YGyro 652  
wait\_async, YHubPort 681  
wait\_async, YHumidity 724  
wait\_async, YLed 755  
wait\_async, YLightSensor 799  
wait\_async, YMagnetometer 845  
wait\_async, YModule 896  
wait\_async, YNetwork 957  
wait\_async, YOsControl 984  
wait\_async, YPower 1031  
wait\_async, YPressure 1074  
wait\_async, YPwmOutput 1116  
wait\_async, YPwmPowerSource 1143  
wait\_async, YQt 1186  
wait\_async, YRealTimeClock 1217  
wait\_async, YRefFrame 1257  
wait\_async, YRelay 1297  
wait\_async, YSensor 1340  
wait\_async, YServo 1379  
wait\_async, YTemperature 1424  
wait\_async, YTilt 1467  
wait\_async, YVoc 1510  
wait\_async, YVoltage 1553  
wait\_async, YVSource 1589  
wait\_async, YWakeUpMonitor 1627  
wait\_async, YWakeUpSchedule 1669  
wait\_async, YWatchdog 1718  
wait\_async, YWireless 1751  
wakeUp, YWakeUpMonitor 1628  
WakeUpMonitor 1591  
WakeUpSchedule 1630  
Watchdog 1671  
Wireless 1720

## Y

YAccelerometer 34-76  
YAnButton 80-118  
YAPI 14-30  
YCarbonDioxide 122-161  
yCheckLogicalName 14  
YColorLed 164-194  
YCompass 198-238  
YCurrent 242-281  
YDataLogger 284-316

YDataRun 318-326  
YDataSet 329-339  
YDataStream 342-354  
YDigitalIO 358-402  
yDisableExceptions 15  
YDisplay 406-453  
YDisplayLayer 456-485  
YDualPower 488-514  
yEnableExceptions 16  
YFiles 517-547  
yFindAccelerometer 34  
yFindAnButton 80  
yFindCarbonDioxide 122  
yFindColorLed 164  
yFindCompass 198  
yFindCurrent 242  
yFindDataLogger 284  
yFindDigitalIO 358  
yFindDisplay 406  
yFindDualPower 488  
yFindFiles 517  
yFindGenericSensor 551  
yFindGyro 601  
yFindHubPort 655  
yFindHumidity 685  
yFindLed 727  
yFindLightSensor 759  
yFindMagnetometer 803  
yFindModule 855  
yFindNetwork 901  
yFindOsControl 960  
yFindPower 988  
yFindPressure 1035  
yFindPwmOutput 1078  
yFindPwmPowerSource 1119  
yFindQt 1147  
yFindRealTimeClock 1189  
yFindRefFrame 1221  
yFindRelay 1261  
yFindSensor 1301  
yFindServo 1344  
yFindTemperature 1383  
yFindTilt 1428  
yFindVoc 1471  
yFindVoltage 1514  
yFindVSource 1556  
yFindWakeUpMonitor 1593  
yFindWakeUpSchedule 1632  
yFindWatchdog 1673  
yFindWireless 1721  
yFirstAccelerometer 35  
yFirstAnButton 81  
yFirstCarbonDioxide 123  
yFirstColorLed 165  
yFirstCompass 199  
yFirstCurrent 243  
yFirstDataLogger 285  
yFirstDigitalIO 359  
yFirstDisplay 407  
yFirstDualPower 489  
yFirstFiles 518  
yFirstGenericSensor 552  
yFirstGyro 602  
yFirstHubPort 656  
yFirstHumidity 686  
yFirstLed 728  
yFirstLightSensor 760  
yFirstMagnetometer 804  
yFirstModule 856  
yFirstNetwork 902  
yFirstOsControl 961  
yFirstPower 989  
yFirstPressure 1036  
yFirstPwmOutput 1079  
yFirstPwmPowerSource 1120  
yFirstQt 1148  
yFirstRealTimeClock 1190  
yFirstRefFrame 1222  
yFirstRelay 1262  
yFirstSensor 1302  
yFirstServo 1345  
yFirstTemperature 1384  
yFirstTilt 1429  
yFirstVoc 1472  
yFirstVoltage 1515  
yFirstVSource 1557  
yFirstWakeUpMonitor 1594  
yFirstWakeUpSchedule 1633  
yFirstWatchdog 1674  
yFirstWireless 1722  
yFreeAPI 17  
YGenericSensor 551-597  
yGetAPIVersion 18  
yGetTickCount 19  
YGyro 601-652  
yHandleEvents 20  
YHubPort 655-681  
YHumidity 685-724  
yInitAPI 21  
YLed 727-755  
YLightSensor 759-799  
YMagnetometer 803-845  
YMeasure 847-851  
YModule 855-896  
YNetwork 901-957  
Yocto-Demo 3  
Yocto-hub 654  
YOscControl 960-984  
YPower 988-1031  
yPreregisterHub 22  
YPressure 1035-1074  
YPwmOutput 1078-1116  
YPwmPowerSource 1119-1143  
YQt 1147-1186  
YRealTimeClock 1189-1217  
YRefFrame 1221-1257  
yRegisterDeviceArrivalCallback 23  
yRegisterDeviceRemovalCallback 24

yRegisterHub 25  
YRelay 1261-1297  
YSensor 1301-1340  
YServo 1344-1379  
ySetTimeout 26  
ySleep 27  
YTemperature 1383-1424  
YTilt 1428-1467  
yUnregisterHub 28

yUpdateDeviceList 29  
yUpdateDeviceList\_async 30  
YVoc 1471-1510  
YVoltage 1514-1553  
YVSource 1556-1589  
YWakeUpMonitor 1593-1628  
YWakeUpSchedule 1632-1669  
YWatchdog 1673-1718  
YWireless 1721-1751