



## Référence de l'API C# .NET



# Table des matières

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Utilisation du Yocto-Demo en C# .....</b>	<b>3</b>
2.1. Installation .....	3
2.2. Utilisation l'API yoctopuce dans un projet Visual C# .....	3
2.3. Contrôle de la fonction Led .....	4
2.4. Contrôle de la partie module .....	6
2.5. 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 .....	39
3.3. Interface de la fonction AnButton .....	85
3.4. Interface de la fonction CarbonDioxide .....	127
3.5. Interface de la fonction ColorLed .....	170
3.6. Interface de la fonction Compass .....	203
3.7. Interface de la fonction Current .....	247
3.8. Interface de la fonction DataLogger .....	290
3.9. Séquence de données mise en forme .....	325
3.10. Séquence de données enregistrées .....	335
3.11. Séquence de données enregistrées brute .....	348
3.12. Interface de la fonction DigitalIO .....	363
3.13. Interface de la fonction Display .....	411
3.14. Interface des objets DisplayLayer .....	462
3.15. Interface de contrôle de l'alimentation .....	494
3.16. Interface de la fonction Files .....	523
3.17. Interface de la fonction GenericSensor .....	556
3.18. Interface de la fonction Gyro .....	606
3.19. Interface d'un port de Yocto-hub .....	661
3.20. Interface de la fonction Humidity .....	690
3.21. Interface de la fonction Led .....	733
3.22. Interface de la fonction LightSensor .....	764
3.23. Interface de la fonction Magnetometer .....	808
3.24. Valeur mesurée .....	854

3.25. Interface de contrôle du module .....	860
3.26. Interface de la fonction Network .....	906
3.27. contrôle d'OS .....	967
3.28. Interface de la fonction Power .....	994
3.29. Interface de la fonction Pressure .....	1041
3.30. Interface de la fonction Pwm .....	1084
3.31. Interface de la fonction PwmPowerSource .....	1126
3.32. Interface du quaternion .....	1153
3.33. Interface de la fonction Horloge Temps Real .....	1196
3.34. Configuration du référentiel .....	1227
3.35. Interface de la fonction Relay .....	1267
3.36. Interface des fonctions de type senseur .....	1307
3.37. Interface de la fonction Servo .....	1350
3.38. Interface de la fonction Temperature .....	1389
3.39. Interface de la fonction Tilt .....	1434
3.40. Interface de la fonction Voc .....	1477
3.41. Interface de la fonction Voltage .....	1520
3.42. Interface de la fonction Source de tension .....	1563
3.43. Interface de la fonction WakeUpMonitor .....	1599
3.44. Interface de la fonction WakeUpSchedule .....	1638
3.45. Interface de la fonction Watchdog .....	1679
3.46. Interface de la fonction Wireless .....	1728
<b>Index .....</b>	<b>1761</b>

# 1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie C# .NET 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 C#

C# (prononcez C-Sharp) est un langage orienté objet promu par Microsoft qui n'est pas sans rappeler Java. Tout comme Visual Basic et Delphi, il permet de créer des applications Windows relativement facilement. Tous les exemples et les modèles de projet sont testés avec Microsoft C# 2010 Express, disponible gratuitement sur le site de Microsoft<sup>1</sup>.

### 2.1. Installation

Téléchargez la librairie Yoctopuce pour Visual C# depuis le site web de Yoctopuce<sup>2</sup>. Il n'y a pas de programme d'installation, copiez simplement le contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire Sources. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual C# 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

### 2.2. Utilisation l'API yoctopuce dans un projet Visual C#

La librairie Yoctopuce pour Visual C# .NET se présente sous la forme d'une DLL et de fichiers sources en Visual C#. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules<sup>3</sup>. Les fichiers sources en Visual C# gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .cs du répertoire Sources pour créer un projet gérant des modules Yoctopuce.

#### Configuration d'un projet Visual C#

Les indications ci-dessous sont fournies pour Visual Studio express 2010, mais la procédure est semblable pour les autres versions.

Commencez par créer votre projet, puis depuis le panneau **Explorateur de solutions** effectuez un clic droit sur votre projet, et choisissez **Ajouter** puis **Elément existant**.

Une fenêtre de sélection de fichiers apparaît: sélectionnez le fichier `yocto_api.cs` et les fichiers correspondant aux fonctions des modules Yoctopuce que votre projet va gérer. Dans le doute, vous pouvez aussi sélectionner tous les fichiers.

---

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

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

<sup>3</sup> Les sources de cette DLL sont disponibles dans l'API C++

Vous avez alors le choix entre simplement ajouter ces fichiers à votre projet, ou les ajouter en tant que lien (le bouton **Ajouter** est en fait un menu déroulant). Dans le premier cas, Visual Studio va copier les fichiers choisis dans votre projet, dans le second Visual Studio va simplement garder un lien sur les fichiers originaux. Il est recommandé d'utiliser des liens, une éventuelle mise à jour de la librairie sera ainsi beaucoup plus facile.

Ensuite, ajoutez de la même manière la dll yapi.dll, qui se trouve dans le répertoire Sources/dll<sup>4</sup>. Puis depuis la fenêtre **Explorateur de solutions**, effectuez un clic droit sur la DLL, choisissez **Propriété** et dans le panneau **Propriétés**, mettez l'option **Copier dans le répertoire de sortie à toujours copier**. Vous êtes maintenant prêt à utiliser vos modules Yoctopuce depuis votre environnement Visual Studio.

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

## 2.3. Contrôle de la fonction Led

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

```
[...]
string errmsg = "";
YLed led;

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

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

Voyons maintenant en détail ce que font ces quelques lignes.

### YAPI.RegisterHub

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

### YLed.FindLed

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

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

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

---

<sup>4</sup> Pensez à changer le filtre de la fenêtre de sélection de fichiers, sinon la DLL n'apparaîtra pas

## isOnline

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

## set\_power

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

## Un exemple réel

Lancez Visual C# et ouvrez le projet exemple correspondant, fourni 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.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void usage()
        {
            string execname = System.AppDomain.CurrentDomain.FriendlyName;
            Console.WriteLine(execname + " <serial_number> [ on | off ]");
            Console.WriteLine(execname + " <logical_name> [ on | off ]");
            Console.WriteLine(execname + " any [ on | off ]");
            System.Threading.Thread.Sleep(2500);
            Environment.Exit(0);
        }

        static void Main(string[] args)
        {
            string errmsg = "";
            string target;
            YLed led;
            string on_off;

            if (args.Length < 2) usage();
            target = args[0].ToUpper();
            on_off = args[1].ToUpper();

            if (YAPI.RegisterHub("usb", ref errmsg) != YAPI.SUCCESS)
            {
                Console.WriteLine("RegisterHub error: " + errmsg);
                Environment.Exit(0);
            }

            if (target == "ANY")
            {
                led = YLed.FirstLed();
                if (led == null)
                {
                    Console.WriteLine("No module connected (check USB cable) ");
                    Environment.Exit(0);
                }
            }
            else led = YLed.FindLed(target + ".led");

            if (led.isOnline())
            {
                if (on_off == "ON") led.set_power(YLed.POWER_ON); else led.set_power(YLed.POWER_OFF);
            }
            else Console.WriteLine("Module not connected (check identification and USB cable)");
        }
    }
}
```

```

    }
}

```

## 2.4. Contrôle de la partie module

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

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void usage()
        { string execname = System.AppDomain.CurrentDomain.FriendlyName;
            Console.WriteLine("Usage:");
            Console.WriteLine(execname+ " <serial or logical name> [ON/OFF]");
            System.Threading.Thread.Sleep(2500);
            Environment.Exit(0);
        }

        static void Main(string[] args)
        {
            YModule m;
            string errmsg = "";

            if (YAPI.RegisterHub("usb", ref errmsg) != YAPI.SUCCESS)
            {
                Console.WriteLine("RegisterHub error: " + errmsg);
                Environment.Exit(0);
            }

            if (args.Length < 1) usage();

            m = YModule.FindModule(args[0]); // use serial or logical name

            if (m.isOnline())
            {
                if (args.Length >= 2)
                {
                    if (args[1].ToUpper() == "ON") { m.set_beacon(YModule.BEACON_ON); }
                    if (args[1].ToUpper() == "OFF") { m.set_beacon(YModule.BEACON_OFF); }
                }

                Console.WriteLine("serial: " + m.get_serialNumber());
                Console.WriteLine("logical name: " + m.get_logicalName());
                Console.WriteLine("luminosity: " + m.get_luminosity().ToString());
                Console.WriteLine("beacon: ");
                if (m.get_beacon() == YModule.BEACON_ON)
                    Console.WriteLine("ON");
                else
                    Console.WriteLine("OFF");
                Console.WriteLine("upTime: " + (m.get_upTime() / 1000).ToString() + " sec");
                Console.WriteLine("USB current: " + m.get_usbCurrent().ToString() + " mA");
                Console.WriteLine("Logs:\r\n" + m.get_lastLogs());

            }
            else
                Console.WriteLine(args[0] + " not connected (check identification and USB cable)");
        }
    }
}

```

Chaque propriété `xxx` du module peut être lue grâce à une méthode du type `YModule.get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `YModule.set_xxxx()`. 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 `YModule.set_xxxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `YModule.saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `YModule.revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void usage()
        { string execname = System.AppDomain.CurrentDomain.FriendlyName;
            Console.WriteLine("Usage:");
            Console.WriteLine("usage: demo <serial or logical name> <new logical name>");
            System.Threading.Thread.Sleep(2500);
            Environment.Exit(0);
        }

        static void Main(string[] args)
        {
            YModule m;
            string errmsg = "";
            string newname;

            if (args.Length != 2) usage();

            if (YAPI.RegisterHub("usb", ref errmsg) != YAPI.SUCCESS)
            {
                Console.WriteLine("RegisterHub error: " + errmsg);
                Environment.Exit(0);
            }

            m = YModule.FindModule(args[0]); // use serial or logical name

            if (m.isOnline())
            {
                newname = args[1];
                if (!YAPI.CheckLogicalName(newname))
                {
                    Console.WriteLine("Invalid name (" + newname + ")");
                    Environment.Exit(0);
                }

                m.set_logicalName(newname);
                m.saveToFlash(); // do not forget this

                Console.Write("Module: serial= " + m.get_serialNumber());
                Console.WriteLine(" / name= " + m.get_logicalName());
            }
            else
                Console.Write("not connected (check identification and USB cable");
        }
    }
}
```

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

liée à la technologie employée par le micro-processeur du module se situe aux alentour de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

## Enumeration des modules

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            YModule m;
            string errmsg = "";

            if (YAPI.RegisterHub("usb", ref errmsg) != YAPI.SUCCESS)
            {
                Console.WriteLine("RegisterHub error: " + errmsg);
                Environment.Exit(0);
            }

            Console.WriteLine("Device list");
            m = YModule.FirstModule();
            while (m!=null)
            {
                Console.WriteLine(m.get_serialNumber() + " (" + m.get_productName() + ")");
                m = m.nextModule();
            }
        }
    }
}
```

## 2.5. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `yDisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur 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 demandant à l'objet qui a renvoyé 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()YAPI.CheckLogicalName()**

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

```
js function yCheckLogicalName( name)
nodejs function CheckLogicalName( name)
php function yCheckLogicalName( $name)
cpp bool yCheckLogicalName( const string& name)
m BOOL yCheckLogicalName( NSString * name)
pas function yCheckLogicalName( name: string): boolean
vb function yCheckLogicalName( ByVal name As String) As Boolean
cs bool CheckLogicalName( string name)
java boolean CheckLogicalName( String name)
py def CheckLogicalName( name)
```

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

**Paramètres :**

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

**Retourne :**

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

**YAPI.DisableExceptions()****YAPI****yDisableExceptions()YAPI.DisableExceptions()**

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

```
js function yDisableExceptions( )  
node.js function DisableExceptions( )  
php function yDisableExceptions( )  
cpp void yDisableExceptions( )  
m void yDisableExceptions( )  
pas procedure yDisableExceptions( )  
vb procedure yDisableExceptions( )  
cs void DisableExceptions( )  
py def DisableExceptions( )
```

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

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

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

```
js function yEnableExceptions( )
nodejs function EnableExceptions( )
php function yEnableExceptions( )
cpp void yEnableExceptions( )
m void yEnableExceptions( )
pas procedure yEnableExceptions( )
vb procedure yEnableExceptions( )
cs void EnableExceptions( )
py def EnableExceptions( )
```

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

## YAPI.EnableUSBHost() yEnableUSBHost()

YAPI

Cette fonction est utilisée uniquement sous Android.

```
java void EnableUSBHost( Object osContext)
```

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

**Paramètres :**

**osContext** un objet de classe `android.content.Context` (ou une sous-classe)

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

YAPI

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

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

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() yGetAPIVersion()YAPI.GetAPIVersion()

YAPI

Retourne la version de la librairie Yoctopuce utilisée.

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

La version est retornée 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() yGetTickCount()YAPI.GetTickCount()

YAPI

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

```
js function yGetTickCount( )  
nodejs function GetTickCount( )  
php function yGetTickCount( )  
cpp u64 yGetTickCount( )  
m u64 yGetTickCount( )  
pas function yGetTickCount( ):u64  
vb function yGetTickCount( ) As Long  
cs ulong GetTickCount( )  
java long GetTickCount( )  
py def GetTickCount( )
```

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

**Retourne :**

un long entier contenant la valeur du compteur de millisecondes.

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

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

```
js function yHandleEvents( errmsg)
node.js function HandleEvents( errmsg)
php function yHandleEvents( &$errmsg)
cpp YRETCODE yHandleEvents( string& errmsg)
m YRETCODE yHandleEvents( NSError** errmsg)
pas function yHandleEvents( var errmsg: string): integer
vb function yHandleEvents( ByRef errmsg As String) As YRETCODE
cs YRETCODE HandleEvents( ref string errmsg)
java int HandleEvents( )
py def HandleEvents( errmsg=None)
```

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

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

### Paramètres :

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

### Retourne :

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

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

YAPI

Initialise la librairie de programmation de Yoctopuce explicitement.

js	function <b>yInitAPI( mode, errmsg)</b>
nodejs	function <b>InitAPI( mode, errmsg)</b>
php	function <b>yInitAPI( \$mode, &amp;\$errmsg)</b>
cpp	<b>YRETCODE yInitAPI( int mode, string&amp; errmsg)</b>
m	<b>YRETCODE yInitAPI( int mode, NSError** errmsg)</b>
pas	function <b>yInitAPI( mode: integer, var errmsg: string): integer</b>
vb	function <b>yInitAPI( ByVal mode As Integer, ByRef errmsg As String) As Integer</b>
cs	<b>int InitAPI( int mode, ref string errmsg)</b>
java	<b>int InitAPI( int mode)</b>
py	<b>def InitAPI( mode, errmsg=None)</b>

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

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

### Paramètres :

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

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

### Retourne :

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

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

YAPI

Alternative plus tolérante à RegisterHub().

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

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

### Paramètres :

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

### Retourne :

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

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

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

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

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

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

**Paramètres :**

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

## YAPI.RegisterDeviceRemovalCallback() yRegisterDeviceRemovalCallback() YAPI.RegisterDeviceRemovalCallback()

YAPI

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

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

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()YAPI.RegisterHub()

YAPI

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

```

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

```

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

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

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

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

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

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

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

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

### Paramètres :

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

### Retourne :

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

### **3. Reference**

---

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

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

YAPI

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

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

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

### Paramètres :

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

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

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

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

Utile pour débugger le fonctionnement de l'API.

**Paramètres :**

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

## YAPI.SelectArchitecture() ySelectArchitecture()

YAPI

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

**PY** `def SelectArchitecture( arch)`

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

### Paramètres :

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

### Retourne :

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

## YAPI.SetDelegate() ySetDelegate()

YAPI

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

m void **ySetDelegate( id object)**

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

### Paramètres :

**object** un objet qui soit se conformer au protocole `YAPIDelegate`, ou `nil`

## YAPI.SetTimeout() ySetTimeout()

YAPI

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

```
js   function ySetTimeout( callback, ms_timeout, arguments )
nodejs function SetTimeout( 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()YAPI.Sleep()

YAPI

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

```
js function ySleep( ms_duration, errmsg)
node.js function Sleep( ms_duration, errmsg)
php function ySleep( $ms_duration, &$errmsg)
cpp YRETCODE ySleep( unsigned ms_duration, string& errmsg)
m YRETCODE ySleep( unsigned ms_duration, NSError ** errmsg)
pas function ySleep( ms_duration: integer, var errmsg: string): integer
vb function ySleep( ByVal ms_duration As Integer,
                   ByRef errmsg As String) As Integer
cs int Sleep( int ms_duration, ref string errmsg)
java int Sleep( long ms_duration)
py def Sleep( ms_duration, errmsg=None)
```

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

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

### Paramètres :

**ms\_duration** un entier correspondant à la durée de la pause, en millisecondes  
**errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

### Retourne :

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

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

Relance une détection des hubs réseau.

cpp	<b>YRETCODE yTriggerHubDiscovery( string&amp; errmsg)</b>
m	<b>+ (YRETCODE) yTriggerHubDiscovery : (NSError**) errmsg</b>
pas	<b>function yTriggerHubDiscovery( var errmsg: string): integer</b>
vb	<b>function yTriggerHubDiscovery( ByRef errmsg As String) As Integer</b>
cs	<b>int TriggerHubDiscovery( ref string errmsg)</b>
java	<b>int TriggerHubDiscovery( )</b>
py	<b>def TriggerHubDiscovery( errmsg=None)</b>

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

**Paramètres :**

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

**Retourne :**

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

## YAPI.UnregisterHub() yUnregisterHub()YAPI.UnregisterHub()

YAPI

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

```
js function yUnregisterHub( url)
nodejs function UnregisterHub( url)
php function yUnregisterHub( $url)
cpp void yUnregisterHub( const string& url)
m void yUnregisterHub( NSString * url)
pas procedure yUnregisterHub( url: string)
vb procedure yUnregisterHub( ByVal url As String)
cs void UnregisterHub( string url)
java void UnregisterHub( String url)
py def UnregisterHub( url)
```

### Paramètres :

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

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

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

<code>js</code>	<code>function yUpdateDeviceList( errmsg)</code>
<code>node.js</code>	<code>function UpdateDeviceList( errmsg)</code>
<code>php</code>	<code>function yUpdateDeviceList( &amp;\$errmsg)</code>
<code>cpp</code>	<code>YRETCODE yUpdateDeviceList( string&amp; errmsg)</code>
<code>m</code>	<code>YRETCODE yUpdateDeviceList( NSError** errmsg)</code>
<code>pas</code>	<code>function yUpdateDeviceList( var errmsg: string): integer</code>
<code>vb</code>	<code>function yUpdateDeviceList( ByRef errmsg As String) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE UpdateDeviceList( ref string errmsg)</code>
<code>java</code>	<code>int UpdateDeviceList( )</code>
<code>py</code>	<code>def UpdateDeviceList( errmsg=None)</code>

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()

YAPI

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

```
js function yUpdateDeviceList_async( callback, context)
node.js function UpdateDeviceList_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 extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### yFindAccelerometer(func)

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

#### yFirstAccelerometer()

Commence l'énumération des accéléromètres accessibles par la librairie.

### Méthodes des objets YAccelerometer

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

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

#### accelerometer→describe()

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

#### accelerometer→get\_advertisedValue()

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

#### accelerometer→get\_currentRawValue()

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

#### accelerometer→get\_currentValue()

Retourne la valeur actuelle de l'accélération.

#### accelerometer→get\_errorMessage()

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

#### accelerometer→get\_errorType()

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

#### accelerometer→get\_friendlyName()

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

#### accelerometer→get\_functionDescriptor()

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

#### accelerometer→get\_functionId()

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

#### accelerometer→get\_hardwareId()

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

### 3. Reference

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

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

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

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

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

Retourne le nom logique de l'accéléromètre.

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

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

#### **accelerometer→get\_module()**

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

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

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

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

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

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

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

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

Retourne la résolution des valeurs mesurées.

#### **accelerometer→get\_unit()**

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

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

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

#### **accelerometer→get\_xValue()**

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

#### **accelerometer→get\_yValue()**

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

#### **accelerometer→get\_zValue()**

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

#### **accelerometer→isOnline()**

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

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

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

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

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

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

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

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

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

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

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

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

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

**accelerometer→registerValueCallback(callback)**

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

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

Modifie la mémoire de valeur maximale observée.

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

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

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

Modifie le nom logique de l'accéléromètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**accelerometer→set(userData)**

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

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

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

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

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

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

L'identifiant peut être spécifié sous plusieurs formes:

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

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

**Paramètres :**

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

**Retourne :**

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

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

Commence l'énumération des accéléromètres accessibles par la librairie.

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

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

**Retourne :**

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

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

YAccelerometer

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

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php   function calibrateFromPoints( $rawValues, $refValues)
cpp    int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)
m     -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YAccelerometer target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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).

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

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

**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.

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

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'accélération

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

**accelerometer→getErrorMessage()**  
**accelerometer→errorMessage()**  
**accelerometer.getErrorMessage()****YAccelerometer**

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

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

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.

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

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.

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

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.

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

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

**Retourne :**

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

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

**YAccelerometer**

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

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

Par exemple `relay1`.

**Retourne :**

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

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

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

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

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.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YAccelerometer <b>target get_highestValue</b>

**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.

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

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

**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.

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

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

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

**Retourne :**

une instance de `YModule`

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

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

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

**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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YAccelerometer target get_recordedData startTime endTime

```

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

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

#### Paramètres :

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

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

#### Retourne :

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

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

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

**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.

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

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

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

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.

```
js function get_xValue( )  
node.js function get_xValue( )  
php function get_xValue( )  
cpp double get_xValue( )  
m -(double) xValue  
pas function get_xValue( ): double  
vb function get_xValue( ) As Double  
cs double get_xValue( )  
java double get_xValue( )  
py def get_xValue( )  
cmd YAccelerometer target 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.

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

**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.

```
js function get_zValue( )  
node.js function get_zValue( )  
php function get_zValue( )  
cpp double get_zValue( )  
m -(double) zValue  
pas function get_zValue( ): double  
vb function get_zValue( ) As Double  
cs double get_zValue( )  
java double get_zValue( )  
py def get_zValue( )  
cmd YAccelerometer target 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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	def <b>isOnline( )</b>

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.

```
js function isOnline_async( callback, context )
nodejs 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.

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

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

**Paramètres :**

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

**Retourne :**

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

## accelerometer→loadCalibrationPoints() accelerometer.loadCalibrationPoints()

YAccelerometer

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

```

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

```

### 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()

## YAccelerometer

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

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextAccelerometer()</b>
node.js	function <b>nextAccelerometer()</b>
php	function <b>nextAccelerometer()</b>
cpp	YAccelerometer * <b>nextAccelerometer()</b>
m	-(YAccelerometer*) <b>nextAccelerometer</b>
pas	function <b>nextAccelerometer()</b> : TYAccelerometer
vb	function <b>nextAccelerometer()</b> As YAccelerometer
cs	YAccelerometer <b>nextAccelerometer()</b>
java	YAccelerometer <b>nextAccelerometer()</b>
py	def <b>nextAccelerometer()</b>

**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.

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

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

### Paramètres :

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

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

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

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

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

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

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

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

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

**Paramètres :**

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

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

**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.

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

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.

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

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

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()****YAccelerometer**

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

```
js function wait_async( callback, context)
node.js 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()**

### 3. Reference

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

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

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

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

**anbutton→set\_logicalName(newval)**

Modifie le nom logique de l'entrée analogique.

**anbutton→set\_sensitivity(newval)**

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

**anbutton→set\_userData(data)**

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

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

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

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

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

<b>js</b>	<code>function yFindAnButton( func)</code>
<b>node.js</b>	<code>function FindAnButton( func)</code>
<b>php</b>	<code>function yFindAnButton( \$func)</code>
<b>cpp</b>	<code>YAnButton* yFindAnButton( const string&amp; func)</code>
<b>m</b>	<code>YAnButton* yFindAnButton( NSString* func)</code>
<b>pas</b>	<code>function yFindAnButton( func: string): TYAnButton</code>
<b>vb</b>	<code>function yFindAnButton( ByVal func As String) As YAnButton</code>
<b>cs</b>	<code>YAnButton FindAnButton( string func)</code>
<b>java</b>	<code>YAnButton FindAnButton( String func)</code>
<b>py</b>	<code>def FindAnButton( 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 l'entrée analogique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAnButton.isOnline()` pour tester si l'entrée analogique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

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

**Retourne :**

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

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

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

js	function <b>yFirstAnButton( )</b>
nodejs	function <b>FirstAnButton( )</b>
php	function <b>yFirstAnButton( )</b>
cpp	<b>YAnButton*</b> <b>yFirstAnButton( )</b>
m	<b>YAnButton*</b> <b>yFirstAnButton( )</b>
pas	function <b>yFirstAnButton( )</b> : TYAnButton
vb	function <b>yFirstAnButton( )</b> As YAnButton
cs	<b>YAnButton</b> <b>FirstAnButton( )</b>
java	<b>YAnButton</b> <b>FirstAnButton( )</b>
py	def <b>FirstAnButton( )</b>

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.

js	function <b>describe()</b>
nodejs	function <b>describe()</b>
php	function <b>describe()</b>
cpp	string <b>describe()</b>
m	-(NSString*) <b>describe</b>
pas	function <b>describe()</b> : string
vb	function <b>describe()</b> As String
cs	string <b>describe()</b>
java	String <b>describe()</b>
py	def <b>describe()</b>

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'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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YAnButton <b>target get_advertisedValue</b>

**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.

```
js function get_analogCalibration( )  
nodejs function get_analogCalibration( )  
php function get_analogCalibration( )  
cpp Y_ANALOGCALIBRATION_enum get_analogCalibration( )  
m -(Y_ANALOGCALIBRATION_enum) analogCalibration  
pas function get_analogCalibration( ): Integer  
vb function get_analogCalibration( ) As Integer  
cs int get_analogCalibration( )  
java int get_analogCalibration( )  
py def get_analogCalibration( )  
cmd YAnButton target 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).

js	function <b>get_calibratedValue( )</b>
node.js	function <b>get_calibratedValue( )</b>
php	function <b>get_calibratedValue( )</b>
cpp	int <b>get_calibratedValue( )</b>
m	-(int) calibratedValue
pas	function <b>get_calibratedValue( )</b> : LongInt
vb	function <b>get_calibratedValue( )</b> As Integer
cs	int <b>get_calibratedValue( )</b>
java	int <b>get_calibratedValue( )</b>
py	def <b>get_calibratedValue( )</b>
cmd	YAnButton <b>target get_calibratedValue</b>

**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).

```
js function get_calibrationMax( )  
nodejs function get_calibrationMax( )  
php function get_calibrationMax( )  
cpp int get_calibrationMax( )  
m -(int) calibrationMax  
pas function get_calibrationMax( ): LongInt  
vb function get_calibrationMax( ) As Integer  
cs int get_calibrationMax( )  
java int get_calibrationMax( )  
py def get_calibrationMax( )  
cmd YAnButton target 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).

js	function <b>get_calibrationMin( )</b>
node.js	function <b>get_calibrationMin( )</b>
php	function <b>get_calibrationMin( )</b>
cpp	int <b>get_calibrationMin( )</b>
m	-(int) calibrationMin
pas	function <b>get_calibrationMin( )</b> : LongInt
vb	function <b>get_calibrationMin( )</b> As Integer
cs	int <b>get_calibrationMin( )</b>
java	int <b>get_calibrationMin( )</b>
py	def <b>get_calibrationMin( )</b>
cmd	YAnButton <b>target get_calibrationMin</b>

**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.

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

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.

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

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.

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

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.

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

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

**Retourne :**

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

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

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

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

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

**anbutton→get.hardwareId()****YAnButton****anbutton→hardwareId()anbutton.get.hardwareId()**

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

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

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.

```
js function get_isPressed( )  
node.js function get_isPressed( )  
php function get_isPressed( )  
cpp Y_ISPRESSED_enum get_isPressed( )  
m -(Y_ISPRESSED_enum) isPressed  
pas function get_isPressed( ): Integer  
vb function get_isPressed( ) As Integer  
cs int get_isPressed( )  
java int get_isPressed( )  
py def get_isPressed( )  
cmd YAnButton target 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).

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

**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).

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

**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.

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

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

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()****YAnButton**

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

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

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

**YAnButton**

Retourne la valeur du compteur d'impulsions.

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

**Retourne :**

un entier représentant la valeur du compteur d'impulsions

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

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

Retourne le timer du compteur d'impulsions (ms)

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

**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 tellequelle (entre 0 et 4095 inclus).

js	function <b>get_rawValue( )</b>
node.js	function <b>get_rawValue( )</b>
php	function <b>get_rawValue( )</b>
cpp	int <b>get_rawValue( )</b>
m	-(int) rawValue
pas	function <b>get_rawValue( ): LongInt</b>
vb	function <b>get_rawValue( ) As Integer</b>
cs	int <b>get_rawValue( )</b>
java	int <b>get_rawValue( )</b>
py	def <b>get_rawValue( )</b>
cmd	<b>YAnButton target get_rawValue</b>

**Retourne :**

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

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

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

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

js	function <b>get_sensitivity()</b>
nodejs	function <b>get_sensitivity()</b>
php	function <b>get_sensitivity()</b>
cpp	int <b>get_sensitivity()</b>
m	-(int) sensitivity
pas	function <b>get_sensitivity()</b> : LongInt
vb	function <b>get_sensitivity()</b> As Integer
cs	int <b>get_sensitivity()</b>
java	int <b>get_sensitivity()</b>
py	def <b>get_sensitivity()</b>
cmd	YAnButton target <b>get_sensitivity</b>

**Retourne :**

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

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

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

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

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

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.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()

YAnButton

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

```
js function isOnline_async( callback, context)
nodejs 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()anbutton.load()****YAnButton**

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

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

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()

YAnButton

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

```
js function load_async( msValidity, callback, context)
nodejs 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()****YAnButton**

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

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

**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.

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

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

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

**Retourne :**

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

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

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

**YAnButton**

Enclenche ou déclenche le procédure de calibration.

```
js function set_analogCalibration( newval)
nodejs function set_analogCalibration( newval)
php function set_analogCalibration( $newval)
cpp int set_analogCalibration( Y_ANALOGCALIBRATION_enum newval)
m -(int) setAnalogCalibration : (Y_ANALOGCALIBRATION_enum) newval
pas function set_analogCalibration( newval: Integer): integer
vb function set_analogCalibration( ByVal newval As Integer) As Integer
cs int set_analogCalibration( int newval)
java int set_analogCalibration( int newval)
py def set_analogCalibration( newval)
cmd YAnButton target 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.

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

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.

```
js function set_calibrationMin( newval)
nodejs function set_calibrationMin( newval)
php function set_calibrationMin( $newval)
cpp int set_calibrationMin( int newval)
m -(int) setCalibrationMin : (int) newval
pas function set_calibrationMin( newval: LongInt): integer
vb function set_calibrationMin( ByVal newval As Integer) As Integer
cs int set_calibrationMin( int newval)
java int set_calibrationMin( int newval)
py def set_calibrationMin( newval)
cmd YAnButton target 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.

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

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

**Paramètres :**

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

```
js function set_sensitivity( newval)
node.js function set_sensitivity( newval)
php function set_sensitivity( $newval)
cpp int set_sensitivity( int newval)
m -(int) setSensitivity : (int) newval
pas function set_sensitivity( newval: LongInt): integer
vb function set_sensitivity( ByVal newval As Integer) As Integer
cs int set_sensitivity( int newval)
java int set_sensitivity( int newval)
py def set_sensitivity( newval)
cmd YAnButton target 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).

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

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()****YAnButton**

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

```
js function wait_async( callback, context)
node.js 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 extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

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

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

#### **yFirstCarbonDioxide()**

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

### Méthodes des objets YCarbonDioxide

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

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

#### **carbondioxide→describe()**

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

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

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

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

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

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

Retourne la valeur actuelle du taux de CO2.

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

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

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

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

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

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

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

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

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

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

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

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

### 3. Reference

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

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

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

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

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

Retourne le nom logique du capteur de CO2.

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

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

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

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

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

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

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

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

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

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

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

Retourne la résolution des valeurs mesurées.

#### **carbon dioxide → get\_unit()**

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

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

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

#### **carbon dioxide → isOnline()**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Modifie la mémoire de valeur maximale observée.

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

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

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

Modifie le nom logique du capteur de CO2.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**carbon dioxide → set\_userData(data)**

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

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

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

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

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

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

L'identifiant peut être spécifié sous plusieurs formes:

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

Cette fonction n'exige pas que 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()****YCarbonDioxide****yFirstCarbonDioxide()****YCarbonDioxide.FirstCarbonDioxide()**

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

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

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

**Retourne :**

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

## carbon dioxide → calibrateFromPoints()

**YCarbonDioxide**

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

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YCarbonDioxide target calibrateFromPoints rawValues refValues

```

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()|carbondioxide.describe()****YCarbonDioxide**

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

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

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)

**carbon dioxide → get\_advertisedValue()**  
**carbon dioxide → advertisedValue()**  
**carbon dioxide.get\_advertisedValue()****YCarbonDioxide**

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

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

js	function <b>get_currentRawValue( )</b>
node.js	function <b>get_currentRawValue( )</b>
php	function <b>get_currentRawValue( )</b>
cpp	double <b>get_currentRawValue( )</b>
m	-(double) currentRawValue
pas	function <b>get_currentRawValue( ): double</b>
vb	function <b>get_currentRawValue( ) As Double</b>
cs	double <b>get_currentRawValue( )</b>
java	double <b>get_currentRawValue( )</b>
py	def <b>get_currentRawValue( )</b>
cmd	YCarbonDioxide <b>target get_currentRawValue</b>

**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**.

**carbon dioxide → get\_currentValue()**  
**carbon dioxide → currentValue()**  
**carbon dioxide.get\_currentValue()****YCarbonDioxide**

Retourne la valeur actuelle du taux de CO<sub>2</sub>.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YCarbonDioxide target get_currentValue
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du taux de CO<sub>2</sub>

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

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

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

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

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

**Retourne :**

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

**carbon dioxide → get\_errorType()**  
**carbon dioxide → errorType()**  
**carbon dioxide.get\_errorType()****YCarbonDioxide**

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

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

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.

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

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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

**Retourne :**

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

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

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

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

Par exemple `relay1`.

**Retourne :**

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

**carbon dioxide → get.hardwareId()**  
**carbon dioxide → hardwareId()**  
**carbon dioxide.get.hardwareId()****YCarbonDioxide**

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

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

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.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YCarbonDioxide <b>target get_highestValue</b>

**Retourne :**

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

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

**carbon dioxide → get\_logFrequency()**  
**carbon dioxide → logFrequency()**  
**carbon dioxide.get\_logFrequency()****YCarbonDioxide**

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

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

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

**carbon dioxide → get\_lowestValue()**  
**carbon dioxide → lowestValue()**  
**carbon dioxide.get\_lowestValue()****YCarbonDioxide**

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

```
js function get_lowestValue( )  
nodejs function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YCarbonDioxide target 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.

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

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

**Retourne :**

une instance de `YModule`

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

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

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

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

## **carbondioxide→recordedData()**

## **carbondioxide.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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs    YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YCarbonDioxide target 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.

**carbon dioxide → get\_reportFrequency()**  
**carbon dioxide → reportFrequency()**  
**carbon dioxide.get\_reportFrequency()****YCarbonDioxide**

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

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

**Retourne :**

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

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

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

Retourne la résolution des valeurs mesurées.

js	function <b>get_resolution( )</b>
node.js	function <b>get_resolution( )</b>
php	function <b>get_resolution( )</b>
cpp	double <b>get_resolution( )</b>
m	-(double) resolution
pas	function <b>get_resolution( ): double</b>
vb	function <b>get_resolution( ) As Double</b>
cs	double <b>get_resolution( )</b>
java	double <b>get_resolution( )</b>
py	<b>def get_resolution( )</b>
cmd	<b>YCarbonDioxide target get_resolution</b>

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

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

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

js	function <b>get_unit()</b>
node.js	function <b>get_unit()</b>
php	function <b>get_unit()</b>
cpp	string <b>get_unit()</b>
m	-(NSString*) unit
pas	function <b>get_unit()</b> : string
vb	function <b>get_unit()</b> As String
cs	string <b>get_unit()</b>
java	String <b>get_unit()</b>
py	def <b>get_unit()</b>
cmd	<b>YCarbonDioxide target get_unit</b>

**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).

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

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.

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

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

js	function <b>isOnline()</b>
node.js	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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

**Retourne :**

true si le capteur de CO2 est joignable, false sinon

## carbondioxide→isOnline\_async()

## YCarbonDioxide

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

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

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

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

### Paramètres :

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

### Retourne :

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

## carbondioxide→loadCalibrationPoints() carbondioxide.loadCalibrationPoints()

**YCarbonDioxide**

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

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)
cmd  YCarbonDioxide target loadCalibrationPoints rawValues refValues

```

### 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()

YCarbonDioxide

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

```
js function load_async( msValidity, callback, context)
nodejs 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.

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

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

js	function <b>nextCarbonDioxide( )</b>
nodejs	function <b>nextCarbonDioxide( )</b>
php	function <b>nextCarbonDioxide( )</b>
cpp	YCarbonDioxide * <b>nextCarbonDioxide( )</b>
m	-(YCarbonDioxide*) <b>nextCarbonDioxide</b>
pas	function <b>nextCarbonDioxide( )</b> : TYCarbonDioxide
vb	function <b>nextCarbonDioxide( )</b> As YCarbonDioxide
cs	YCarbonDioxide <b>nextCarbonDioxide( )</b>
java	YCarbonDioxide <b>nextCarbonDioxide( )</b>
py	def <b>nextCarbonDioxide( )</b>

**Retourne :**

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

## carbon dioxide → registerTimedReportCallback() carbon dioxide.registerTimedReportCallback()

YCarbonDioxide

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

```
js function registerTimedReportCallback( callback )
node.js function registerTimedReportCallback( callback )
php function registerTimedReportCallback( $callback )
cpp int registerTimedReportCallback( YCarbonDioxideTimedReportCallback callback )
m -(int) registerTimedReportCallback : (YCarbonDioxideTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYCarbonDioxideTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback )
java int registerTimedReportCallback( TimedReportCallback callback )
py def registerTimedReportCallback( callback )
```

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

### Paramètres :

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

## carbondioxide→registerValueCallback() carbondioxide.registerValueCallback()

**YCarbonDioxide**

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

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

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.

**carbon dioxide**→**set\_highestValue()**  
**carbon dioxide**→**setHighestValue()**  
**carbon dioxide.set\_highestValue()**

**YCarbonDioxide**

Modifie la mémoire de valeur maximale observée.

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

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

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

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.

**carbon dioxide → set\_logicalName()**  
**carbon dioxide → setLogicalName()**  
**carbon dioxide.set\_logicalName()**

**YCarbonDioxide**

Modifie le nom logique du capteur de CO2.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YCarbonDioxide 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 du capteur de CO2.

**Retourne :**

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

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

**YCarbonDioxide**

Modifie la mémoire de valeur minimale observée.

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

**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.

**carbon dioxide → set\_reportFrequency()**  
**carbon dioxide → setReportFrequency()**  
**carbon dioxide.set\_reportFrequency()****YCarbonDioxide**

Modifie la fréquence de notification périodique des valeurs mesurées.

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

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

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie la résolution des valeurs physique mesurées.

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

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).

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

**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.

```
js function wait_async( callback, context )
nodejs 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>
nodejs var yoctolib = require('yoctolib');
var YColorLed = yoctolib.YColorLed;
php require_once('yocto_colorled.php');
cpp #include "yocto_colorled.h"
m #import "yocto_colorled.h"
pas uses yocto_colorled;
vb yocto_colorled.vb
cs yocto_colorled.cs
java import com.yoctopuce.YoctoAPI.YColorLed;
py from yocto_colorled import *

```

### Fonction globales

#### yFindColorLed(func)

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

#### yFirstColorLed()

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

### Méthodes des objets YColorLed

#### colorled→describe()

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

#### colorled→get\_advertisedValue()

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

#### colorled→get\_errorMessage()

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

#### colorled→get\_errorType()

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

#### colorled→get\_friendlyName()

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

#### colorled→get\_functionDescriptor()

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

#### colorled→get\_functionId()

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

#### colorled→get\_hardwareId()

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

#### colorled→get\_hslColor()

Retourne la couleur HSL courante de la led.

#### colorled→get\_logicalName()

Retourne le nom logique de la led RGB.

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

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

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

```
js function yFindColorLed( func)
node.js function FindColorLed( func)
php function yFindColorLed( $func)
cpp YColorLed* yFindColorLed( const string& func)
m YColorLed* yFindColorLed( NSString* func)
pas function yFindColorLed( func: string): TYColorLed
vb function yFindColorLed( ByVal func As String) As YColorLed
cs YColorLed FindColorLed( string func)
java YColorLed FindColorLed( String func)
py def FindColorLed( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

**Paramètres :**

**func** une chaîne de caractères qui référence la led RGB sans ambiguïté

**Retourne :**

un objet de classe `YColorLed` qui permet ensuite de contrôler la led RGB.

**YColorLed.FirstColorLed()****yFirstColorLed() YColorLed.FirstColorLed()****YColorLed**

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

```
js function yFirstColorLed( )  
nodejs function FirstColorLed( )  
php function yFirstColorLed( )  
cpp YColorLed* yFirstColorLed( )  
m YColorLed* yFirstColorLed( )  
pas function yFirstColorLed( ): TYColorLed  
vb function yFirstColorLed( ) As YColorLed  
cs YColorLed FirstColorLed( )  
java YColorLed FirstColorLed( )  
py def FirstColorLed( )
```

Utiliser la fonction `YColorLed.nextColorLed()` pour itérer sur les autres leds RGB.

**Retourne :**

un pointeur sur un objet `YColorLed`, correspondant à la première led RGB accessible en ligne, ou `null` si il n'y a pas de leds RGB disponibles.

**colorled→describe()colorled.describe()****YColorLed**

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

js	function <b>describe()</b>
nodejs	function <b>describe()</b>
php	function <b>describe()</b>
cpp	string <b>describe()</b>
m	-(NSString*) <b>describe</b>
pas	function <b>describe()</b> : string
vb	function <b>describe()</b> As String
cs	string <b>describe()</b>
java	String <b>describe()</b>
py	def <b>describe()</b>

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

**Retourne :**

une chaîne de caractères décrivant la 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YColorLed <b>target get_advertisedValue</b>

**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.

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

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.

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

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.

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

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.

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

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

**Retourne :**

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

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

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

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

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

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

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

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

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.

```
js function get_hslColor( )  
node.js function get_hslColor( )  
php function get_hslColor( )  
cpp int get_hslColor( )  
m -(int) hslColor  
pas function get_hslColor( ): LongInt  
vb function get_hslColor( ) As Integer  
cs int get_hslColor( )  
java int get_hslColor( )  
py def get_hslColor( )  
cmd YColorLed target 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.

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

**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()**  
**colorled→module()colorled.get\_module()****YColorLed**

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

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

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()****YColorLed**

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

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

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

Retourne la couleur RGB courante de la led.

```
js function get_rgbColor( )  
node.js function get_rgbColor( )  
php function get_rgbColor( )  
cpp int get_rgbColor( )  
m -(int) rgbColor  
pas function get_rgbColor( ): LongInt  
vb function get_rgbColor( ) As Integer  
cs int get_rgbColor( )  
java int get_rgbColor( )  
py def get_rgbColor( )  
cmd YColorLed target 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.

js	function get_rgbColorAtPowerOn( )
node.js	function get_rgbColorAtPowerOn( )
php	function get_rgbColorAtPowerOn( )
cpp	int get_rgbColorAtPowerOn( )
m	-(int) rgbColorAtPowerOn
pas	function get_rgbColorAtPowerOn( ): LongInt
vb	function get_rgbColorAtPowerOn( ) As Integer
cs	int get_rgbColorAtPowerOn( )
java	int get_rgbColorAtPowerOn( )
py	def get_rgbColorAtPowerOn( )
cmd	YColorLed target get_rgbColorAtPowerOn

**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).

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

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.

```

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

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()

## YColorLed

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

```
js   function isOnline_async( callback, context )
nodejs 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.

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

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()

## YColorLed

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

```
js function load_async( msValidity, callback, context)
nodejs 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.

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

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

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

**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.

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

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

### Paramètres :

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

## colorled→rgbMove()colorled.rgbMove()

YColorLed

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

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

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

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.

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

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

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.

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

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

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()

YColorLed

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

```
js function wait_async( callback, context)
node.js 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 retournée par le capteur (sans arrondi ni calibration).

#### compass→get\_currentValue()

Retourne la valeur actuelle du cap relatif.

#### compass→get\_errorMessage()

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

#### compass→get\_errorType()

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

#### compass→get\_friendlyName()

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

#### compass→get\_functionDescriptor()

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

#### compass→get\_functionId()

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

#### compass→get\_hardwareId()

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

### 3. Reference

<b>compass→get_highestValue()</b>	Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.
<b>compass→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>compass→get_logicalName()</b>	Retourne le nom logique du compas.
<b>compass→get_lowestValue()</b>	Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.
<b>compass→get_magneticHeading()</b>	Retourne la direction du nord magnétique, indépendamment du cap configuré.
<b>compass→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>compass→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>compass→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>compass→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>compass→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>compass→get_unit()</b>	Retourne l'unité dans laquelle le cap relatif est exprimée.
<b>compass→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>compass→isOnline()</b>	Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.
<b>compass→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.
<b>compass→load(msValidity)</b>	Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.
<b>compass→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>compass→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.
<b>compass→nextCompass()</b>	Continue l'énumération des compas commencée à l'aide de yFirstCompass( ).
<b>compass→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>compass→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>compass→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.

**compass→set\_logFrequency(newval)**

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

**compass→set\_logicalName(newval)**

Modifie le nom logique du compas.

**compass→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**compass→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**compass→set\_userData(data)**

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

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

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

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

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

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

## YCompass

Commence l'énumération des compas accessibles par la librairie.

js	function <b>yFirstCompass()</b>
nodejs	function <b>FirstCompass()</b>
php	function <b>yFirstCompass()</b>
cpp	YCompass* <b>yFirstCompass()</b>
m	YCompass* <b>yFirstCompass()</b>
pas	function <b>yFirstCompass()</b> : TYCompass
vb	function <b>yFirstCompass()</b> As YCompass
cs	YCompass <b>FirstCompass()</b>
java	YCompass <b>FirstCompass()</b>
py	def <b>FirstCompass()</b>

Utiliser la fonction `YCompass.nextCompass()` pour itérer sur les autres compas.

#### Retourne :

un pointeur sur un objet `YCompass`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YCompass target calibrateFromPoints rawValues refValues

```

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.

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

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YCompass target 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 rentrée par le capteur (sans arrondi ni calibration).

js	function <b>get_currentRawValue( )</b>
node.js	function <b>get_currentRawValue( )</b>
php	function <b>get_currentRawValue( )</b>
cpp	double <b>get_currentRawValue( )</b>
m	-(double) currentRawValue
pas	function <b>get_currentRawValue( ): double</b>
vb	function <b>get_currentRawValue( ) As Double</b>
cs	double <b>get_currentRawValue( )</b>
java	double <b>get_currentRawValue( )</b>
py	def <b>get_currentRawValue( )</b>
cmd	YCompass <b>target get_currentRawValue</b>

**Retourne :**

une valeur numérique représentant la valeur brute rentrée 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.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YCompass target get_currentValue
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du cap relatif

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

**compass→getErrorMessage()  
compass→errorMessage()  
compass.getErrorMessage()****YCompass**

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

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

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.

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

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.

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

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.

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

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

**Retourne :**

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

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

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

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

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.

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

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.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YCompass <b>target get_highestValue</b>

**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.

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

**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.

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

**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.

```
js function get_lowestValue( )  
node.js function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YCompass target 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.

```
js   function get_magneticHeading( )  
node.js function get_magneticHeading( )  
php  function get_magneticHeading( )  
cpp   double get_magneticHeading( )  
m    -(double) magneticHeading  
pas   function get_magneticHeading( ): double  
vb    function get_magneticHeading( ) As Double  
cs    double get_magneticHeading( )  
java  double get_magneticHeading( )  
py    def get_magneticHeading( )  
cmd   YCompass target get_magneticHeading
```

**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.

```
js function get_module()
node.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`

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

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

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

**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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m     -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime
pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs    YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YCompass target get_recordedData startTime endTime

```

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

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

**Paramètres :**

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

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

**Retourne :**

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

**compass→get\_reportFrequency()**  
**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.

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

**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.

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

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

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

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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	def <b>isOnline( )</b>

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()

YCompass

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

```
js   function isOnline_async( callback, context )
nodejs 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.

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

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.

```

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

```

### 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()

## YCompass

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

```
js   function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextCompass()</b>
nodejs	function <b>nextCompass()</b>
php	function <b>nextCompass()</b>
cpp	<b>YCompass * nextCompass()</b>
m	<b>-(YCompass*) nextCompass</b>
pas	function <b>nextCompass()</b> : TYCompass
vb	function <b>nextCompass()</b> As YCompass
cs	<b>YCompass nextCompass()</b>
java	<b>YCompass nextCompass()</b>
py	<b>def nextCompass()</b>

**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.

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

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.

## compass→registerValueCallback() compass.registerValueCallback()

YCompass

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

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

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

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

**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.

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

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

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

**Paramètres :**

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

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

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

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.

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

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

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()

YCompass

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

```
js function wait_async( callback, context)
node.js 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>
node.js	var yoctolib = require('yoctolib');
	var YCurrent = yoctolib.YCurrent;
php	require_once('yocto_current.php');
cpp	#include "yocto_current.h"
m	#import "yocto_current.h"
pas	uses yocto_current;
vb	yocto_current.vb
cs	yocto_current.cs
java	import com.yoctopuce.YoctoAPI.YCurrent;
py	from yocto_current import *

### Fonction globales

#### yFindCurrent(func)

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

#### yFirstCurrent()

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

### Méthodes des objets YCurrent

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

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

#### current→describe()

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

#### current→get\_advertisedValue()

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

#### current→get\_currentRawValue()

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

#### current→get\_currentValue()

Retourne la valeur instantanée du courant.

#### current→get\_errorMessage()

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

#### current→get\_errorType()

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

#### current→get\_friendlyName()

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

#### current→get\_functionDescriptor()

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

#### current→get\_functionId()

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

#### current→get\_hardwareId()

### 3. Reference

Retourne l'identifiant matériel unique du capteur de courant au format SERIAL.FUNCTIONID.
<b>current→get_highestValue()</b> Retourne la valeur maximale observée pour le courant.
<b>current→get_logFrequency()</b> Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>current→get_logicalName()</b> Retourne le nom logique du capteur de courant.
<b>current→get_lowestValue()</b> Retourne la valeur minimale observée pour le courant.
<b>current→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>current→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>current→get_recordedData(startTime, endTime)</b> Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>current→get_reportFrequency()</b> Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>current→get_resolution()</b> Retourne la résolution des valeurs mesurées.
<b>current→get_unit()</b> Retourne l'unité dans laquelle le courant est exprimée.
<b>current→get(userData)</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>current→isOnline()</b> Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.
<b>current→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.
<b>current→load(msValidity)</b> Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.
<b>current→loadCalibrationPoints(rawValues, refValues)</b> Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>current→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.
<b>current→nextCurrent()</b> Continue l'énumération des capteurs de courant commencée à l'aide de yFirstCurrent( ).
<b>current→registerTimedReportCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>current→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>current→set_highestValue(newval)</b> Modifie la mémoire de valeur maximale observée pour le courant.
<b>current→set_logFrequency(newval)</b>

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

**current→set\_logicalName(newval)**

Modifie le nom logique du capteur de courant.

**current→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour le courant.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**current→set\_resolution(newval)**

Modifie la résolution des valeurs mesurées.

**current→set\_userData(data)**

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

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

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

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

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

<b>js</b>	<code>function yFindCurrent( func)</code>
<b>node.js</b>	<code>function FindCurrent( func)</code>
<b>php</b>	<code>function yFindCurrent( \$func)</code>
<b>cpp</b>	<code>YCurrent* yFindCurrent( const string&amp; func)</code>
<b>m</b>	<code>YCurrent* yFindCurrent( NSString* func)</code>
<b>pas</b>	<code>function yFindCurrent( func: string): TYCurrent</code>
<b>vb</b>	<code>function yFindCurrent( ByVal func As String) As YCurrent</code>
<b>cs</b>	<code>YCurrent FindCurrent( string func)</code>
<b>java</b>	<code>YCurrent FindCurrent( String func)</code>
<b>py</b>	<code>def FindCurrent( 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 le capteur de courant soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

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

**Retourne :**

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

## YCurrent.FirstCurrent()

## YCurrent

### yFirstCurrent() YCurrent.FirstCurrent()

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

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

Utiliser la fonction `YCurrent.nextCurrent()` pour itérer sur les autres capteurs de courant.

#### Retourne :

un pointeur sur un objet `YCurrent`, correspondant à le premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

## current→calibrateFromPoints() 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php   function calibrateFromPoints( $rawValues, $refValues)
cpp    int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)
m     -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YCurrent target calibrateFromPoints rawValues refValues

```

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.

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

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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 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).

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

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

js	function <b>get_currentRawValue( )</b>
node.js	function <b>get_currentRawValue( )</b>
php	function <b>get_currentRawValue( )</b>
cpp	double <b>get_currentRawValue( )</b>
m	-(double) currentRawValue
pas	function <b>get_currentRawValue( ): double</b>
vb	function <b>get_currentRawValue( ) As Double</b>
cs	double <b>get_currentRawValue( )</b>
java	double <b>get_currentRawValue( )</b>
py	def <b>get_currentRawValue( )</b>
cmd	<b>YCurrent target get_currentRawValue</b>

**Retourne :**

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

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

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

Retourne la valeur instantanée du courant.

```
js function get_currentValue( )  
node.js function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YCurrent target 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.

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

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()  
current→errorType()current.get\_errorType()****YCurrent**

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

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

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.

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

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.

js	function get_functionDescriptor( )
nodejs	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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

**Retourne :**

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

**current→get\_functionId()****YCurrent****current→functionId()current.get\_functionId()**

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

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

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.

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

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.

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

**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.

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

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

**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.

**js** function **get\_lowestValue( )****node.js** function **get\_lowestValue( )****php** function **get\_lowestValue( )****cpp** double **get\_lowestValue( )****m** -(double) lowestValue**pas** function **get\_lowestValue( ): double****vb** function **get\_lowestValue( ) As Double****cs** double **get\_lowestValue( )****java** double **get\_lowestValue( )****py** def **get\_lowestValue( )****cmd** YCurrent target **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()  
current→module()current.get\_module()****YCurrent**

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

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

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

**Retourne :**

une instance de `YModule`

**current→get\_module\_async()**  
**current→module\_async()****YCurrent**

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

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

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

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

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YCurrent target 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.

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

**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.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YCurrent target 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()**  
**current→unit()current.get\_unit()****YCurrent**

Retourne l'unité dans laquelle le courant est exprimée.

js	function <b>get_unit()</b>
node.js	function <b>get_unit()</b>
php	function <b>get_unit()</b>
cpp	string <b>get_unit()</b>
m	-(NSString*) unit
pas	function <b>get_unit()</b> : string
vb	function <b>get_unit()</b> As String
cs	string <b>get_unit()</b>
java	String <b>get_unit()</b>
py	def <b>get_unit()</b>
cmd	<b>YCurrent target get_unit</b>

**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).

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

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.

```
js function isOnline( )  
nodejs function isOnline( )  
php function isOnline( )  
cpp bool isOnline( )  
m -(BOOL) isOnline  
pas function isOnline( ): boolean  
vb function isOnline( ) As Boolean  
cs bool isOnline( )  
java boolean isOnline( )  
py def isOnline( )
```

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()

## YCurrent

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

```
js   function isOnline_async( callback, context )
nodejs 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.

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

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)
cmd  YCurrent target 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()

YCurrent

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

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextCurrent()</b>
nodejs	function <b>nextCurrent()</b>
php	function <b>nextCurrent()</b>
cpp	YCurrent * <b>nextCurrent()</b>
m	-( <b>YCurrent*</b> ) <b>nextCurrent</b>
pas	function <b>nextCurrent()</b> : TYCurrent
vb	function <b>nextCurrent()</b> As YCurrent
cs	YCurrent <b>nextCurrent()</b>
java	YCurrent <b>nextCurrent()</b>
py	def <b>nextCurrent()</b>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YCurrentTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YCurrentTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYCurrentTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

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

## current→registerValueCallback() current.registerValueCallback()

YCurrent

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

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

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

### Paramètres :

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

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

**YCurrent**

Modifie la mémoire de valeur maximale observée pour le courant.

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

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

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

**Paramètres :**

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

**Retourne :**

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

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

**current→set\_logicalName()****YCurrent****current→setLogicalName()current.set\_logicalName()**

Modifie le nom logique du capteur de courant.

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

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

**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.

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

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

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).

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData)</b>
cpp	void <b>set(userData)</b>
m	-(void) <b>setUserData</b> : (void*) <b>data</b>
pas	procedure <b>set(userData)</b>
vb	procedure <b>set(userData)</b> ByVal <b>data</b> As Object
cs	void <b>set(userData)</b>
java	void <b>set(userData)</b>
py	def <b>set(userData)</b>

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()****YCurrent**

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

```
js function wait_async( callback, context)
nodejs 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>
nodejs var yoctolib = require('yoctolib');
var YDataLogger = yoctolib.YDataLogger;
require_once('yocto_datalogger.php');
php #include "yocto_datalogger.h"
cpp #import "yocto_datalogger.h"
m uses yocto_datalogger;
pas yocto_datalogger.vb
cs yocto_datalogger.cs
java import com.yoctopuce.YoctoAPI.YDataLogger;
py from yocto_datalogger import *

```

### Fonction globales

#### **yFindDataLogger(func)**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

#### **yFirstDataLogger()**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

### Méthodes des objets YDataLogger

#### **datalogger→describe()**

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

#### **datalogger→forgetAllDataStreams()**

Efface tout l'historique des mesures de l'enregistreur de données.

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

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

#### **datalogger→get\_autoStart()**

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

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

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

#### **datalogger→get\_dataSets()**

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

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

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

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

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

#### **datalogger→get\_errorType()**

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

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

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

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

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

#### **datalogger→get\_functionId()**

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

#### **datalogger→get\_hardwareId()**

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

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

Retourne le nom logique de l'enregistreur de données.

#### **datalogger→get\_module()**

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

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

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

#### **datalogger→get\_recording()**

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

#### **datalogger→get\_timeUTC()**

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

#### **datalogger→get\_userData()**

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

#### **datalogger→isOnline()**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

#### **datalogger→load(msValidity)**

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

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

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

#### **datalogger→nextDataLogger()**

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

#### **datalogger→registerValueCallback(callback)**

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

#### **datalogger→set\_autoStart(newval)**

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

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

Modifie le nom logique de l'enregistreur de données.

#### **datalogger→set\_recording(newval)**

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

#### **datalogger→set\_timeUTC(newval)**

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

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

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

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

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

**YDataLogger.FindDataLogger()****YDataLogger****yFindDataLogger()YDataLogger.FindDataLogger()**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

<b>js</b>	<code>function yFindDataLogger( func)</code>
<b>node.js</b>	<code>function FindDataLogger( func)</code>
<b>php</b>	<code>function yFindDataLogger( \$func)</code>
<b>cpp</b>	<code>YDataLogger* yFindDataLogger( string func)</code>
<b>m</b>	<code>+ (YDataLogger*) yFindDataLogger : (NSString*) func</code>
<b>pas</b>	<code>function yFindDataLogger( func: string): TYDataLogger</code>
<b>vb</b>	<code>function yFindDataLogger( ByVal func As String) As YDataLogger</code>
<b>cs</b>	<code>YDataLogger FindDataLogger( string func)</code>
<b>java</b>	<code>YDataLogger FindDataLogger( String func)</code>
<b>py</b>	<code>def FindDataLogger( 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 l'enregistreur de données soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnline()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

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

**Retourne :**

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

**YDataLogger.FirstDataLogger()****YDataLogger****yFirstDataLogger()YDataLogger.FirstDataLogger()**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

js	function <b>yFirstDataLogger( )</b>
nodejs	function <b>FirstDataLogger( )</b>
php	function <b>yFirstDataLogger( )</b>
cpp	YDataLogger* <b>yFirstDataLogger( )</b>
m	YDataLogger* <b>yFirstDataLogger( )</b>
pas	function <b>yFirstDataLogger( ): TYDataLogger</b>
vb	function <b>yFirstDataLogger( ) As YDataLogger</b>
cs	YDataLogger <b>FirstDataLogger( )</b>
java	YDataLogger <b>FirstDataLogger( )</b>
py	def <b>FirstDataLogger( )</b>

Utiliser la fonction `YDataLogger.nextDataLogger()` pour itérer sur les autres enregistreurs de données.

**Retourne :**

un pointeur sur un objet `YDataLogger`, correspondant à le premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

**datalogger→describe()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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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.

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

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

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

**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.

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

**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.

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

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).

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

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.

js	function <b>get_errorMessage( )</b>
nodejs	function <b>getErrorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	<b>String get_errorMessage( )</b>
py	<b>def get_errorMessage( )</b>

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.

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

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.

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

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.

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

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

**Retourne :**

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

**datalogger→get\_functionId()****YDataLogger****datalogger→functionId()datalogger.get\_functionId()**

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

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

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.

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

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.

js	function <b>get_logicalName( )</b>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( )</b> : string
vb	function <b>get_logicalName( )</b> As String
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	YDataLogger <b>target get_logicalName</b>

**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.

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

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()****YDataLogger**

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

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

```
js function get_recording( )
node.js function get_recording( )
php function get_recording( )
cpp Y_RECORDING_enum get_recording( )
m -(Y_RECORDING_enum) recording
pas function get_recording( ): Integer
vb function get_recording( ) As Integer
cs int get_recording( )
java int get_recording( )
py def get_recording( )
cmd YDataLogger target 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.

js	function <b>get_timeUTC()</b>
nodejs	function <b>get_timeUTC()</b>
php	function <b>get_timeUTC()</b>
cpp	s64 <b>get_timeUTC()</b>
m	-(s64) timeUTC
pas	function <b>get_timeUTC()</b> : int64
vb	function <b>get_timeUTC()</b> As Long
cs	long <b>get_timeUTC()</b>
java	long <b>get_timeUTC()</b>
py	def <b>get_timeUTC()</b>
cmd	YDataLogger target <b>get_timeUTC</b>

**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).

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

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.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()****YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

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

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()****YDataLogger**

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

**js** `function load_async( msValidity, callback, context)`  
**nodejs** `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()`.

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

**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.

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

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

<b>js</b>	<code>function set_autoStart( newval)</code>
<b>node.js</b>	<code>function set_autoStart( newval)</code>
<b>php</b>	<code>function set_autoStart( \$newval)</code>
<b>cpp</b>	<code>int set_autoStart( Y_AUTOSTART_enum newval)</code>
<b>m</b>	<code>-(int) setAutoStart : (Y_AUTOSTART_enum) newval</code>
<b>pas</b>	<code>function set_autoStart( newval: Integer): integer</code>
<b>vb</b>	<code>function set_autoStart( ByVal newval As Integer) As Integer</code>
<b>cs</b>	<code>int set_autoStart( int newval)</code>
<b>java</b>	<code>int set_autoStart( int newval)</code>
<b>py</b>	<code>def set_autoStart( newval)</code>
<b>cmd</b>	<code>YDataLogger target set_autoStart newval</code>

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.

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

js	function <b>set_recording( newval)</b>
node.js	function <b>set_recording( newval)</b>
php	function <b>set_recording( \$newval)</b>
cpp	int <b>set_recording( Y_RECORDING_enum newval)</b>
m	-(int) <b>setRecording : (Y_RECORDING_enum) newval</b>
pas	function <b>set_recording( newval: Integer): integer</b>
vb	function <b>set_recording( ByVal newval As Integer) As Integer</b>
cs	int <b>set_recording( int newval)</b>
java	int <b>set_recording( int newval)</b>
py	def <b>set_recording( newval)</b>
cmd	YDataLogger <b>target set_recording newval</b>

**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.

```
js function set_timeUTC( newval)
node.js function set_timeUTC( newval)
php function set_timeUTC( $newval)
cpp int set_timeUTC( s64 newval)
m -(int) setTimeUTC : (s64) newval
pas function set_timeUTC( newval: int64): integer
vb function set_timeUTC( ByVal newval As Long) As Integer
cs int set_timeUTC( long newval)
java int set_timeUTC( long newval)
py def set_timeUTC( newval)
cmd YDataLogger target 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).

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

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()****YDataLogger**

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

```
js function wait_async( callback, context)
node.js 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>
node.js var yoctolib = require('yoctolib');
          var YDataLogger = yoctolib.YDataLogger;
php require_once('yocto_datalogger.php');
cpp #include "yocto_datalogger.h"
m #import "yocto_datalogger.h"
pas uses yocto_datalogger;
vb yocto_datalogger.vb
cs yocto_datalogger.cs
java import com.yoctopuce.YoctoAPI.YDataLogger;
py from yocto_datalogger import *

```

### Méthodes des objets YDataRun

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

Retourne la valeur moyenne des mesures observées au moment choisi.

#### **datarun→get\_duration()**

Retourne la durée (en secondes) du Run.

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

Retourne la valeur maximale des mesures observées au moment choisi.

#### **datarun→get\_measureNames()**

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

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

Retourne la valeur minimale des mesures observées au moment choisi.

#### **datarun→get\_startTimeUTC()**

Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### **datarun→get\_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

#### **datarun→get\_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

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

Change l'intervalle de temps représenté par chaque valeur de ce run.

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

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

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

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()****YDataRun**

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

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

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

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()****YDataRun**

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-mêmes sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Méthodes des objets YDataSet

#### `dataset→get_endTimeUTC()`

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### `dataset→get_functionId()`

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

#### `dataset→get_hardwareId()`

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

#### `dataset→get_measures()`

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

#### `dataset→get_preview()`

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

#### `dataset→get_progress()`

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

#### `dataset→get_startTimeUTC()`

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### `dataset→get_summary()`

Retourne un objet YMeasure résumant tout le YDataSet.

#### `dataset→get_unit()`

### **3. Reference**

---

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

#### **dataset→loadMore()**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

#### **dataset→loadMore\_async(callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**dataset→get\_endTimeUTC()****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).

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

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.

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

Par exemple `temperature1`.

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: `temperature1`)

**dataset→get\_hardwareId()****YDataSet****dataset→hardwareId()dataset.get\_hardwareId()**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

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

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple THRMCPL1-123456.temperature1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: THRMCPL1-123456.temperature1)

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

**dataset→get\_measures()****YDataSet****dataset→measures()dataset.get\_measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

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

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.

js	<code>function get_preview( )</code>
nodejs	<code>function get_preview( )</code>
php	<code>function get_preview( )</code>
cpp	<code>vector&lt;YMeasure&gt; get_preview( )</code>
m	<code>-(NSMutableArray*) preview</code>
pas	<code>function get_preview( ): TYMeasureArray</code>
vb	<code>function get_preview( ) As List</code>
cs	<code>List&lt;YMeasure&gt; get_preview( )</code>
java	<code>ArrayList&lt;YMeasure&gt; get_preview( )</code>
py	<code>def get_preview( )</code>

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.

```
js function get_progress( )
node.js function get_progress( )
php function get_progress( )
cpp int get_progress( )
m -(int) progress
pas function get_progress( ): LongInt
vb function get_progress( ) As Integer
cs int get_progress( )
java int get_progress( )
py def get_progress( )
```

A l'instanciation de l'objet par la fonction `get_dataSet()`, l'avancement est nul. Au fur et à mesure des appels à `loadMore()`, l'avancement progresse pour atteindre la valeur 100 lorsque toutes les mesures ont été chargées.

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées.

**dataset→getStartTimeUTC()****YDataSet****dataset→startTimeUTC()dataset.getStartTimeUTC()**

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).

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

```
js function get_summary( )
node.js function get_summary( )
php function get_summary( )
cpp YMeasure get_summary( )
m -(YMeasure*) summary
pas function get_summary( ): TYMeasure
vb function get_summary( ) As YMeasure
cs YMeasure get_summary( )
java YMeasure get_summary( )
py def get_summary( )
```

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

**Retourne :**

un objet YMeasure

**dataset→get\_unit()  
dataset→unit()dataset.get\_unit()****YDataSet**

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

js	function <b>get_unit( )</b>
nodejs	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>

**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.

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

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

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

**dataset→loadMore\_async()****YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
js function loadMore_async( callback, context)
nodejs 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>
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 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()`

Retourne le temps de départ relatif de la séquence (en secondes).

**datastream→getStartTimeUTC()**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

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

**YDataStream**

Retourne la moyenne des valeurs observées durant cette séquence.

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

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 contenues dans la séquence.

js	function <b>get_columnCount( )</b>
node.js	function <b>get_columnCount( )</b>
php	function <b>get_columnCount( )</b>
cpp	<b>int get_columnCount( )</b>
m	<b>-(int) columnCount</b>
pas	function <b>get_columnCount( ): LongInt</b>
vb	function <b>get_columnCount( ) As Integer</b>
cs	<b>int get_columnCount( )</b>
java	<b>int get_columnCount( )</b>
py	<b>def get_columnCount( )</b>

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames( )`.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre de colonnes.

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

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

**YDataStream**

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

```
js function get_columnNames( )  
nodejs function get_columnNames( )  
php function get_columnNames( )  
cpp vector<string> get_columnNames( )  
m -(NSMutableArray*) columnNames  
pas function get_columnNames( ): TStringArray  
vb function get_columnNames( ) As List  
cs List<string> get_columnNames( )  
java ArrayList<String> get_columnNames( )  
py def get_columnNames( )
```

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure. Pour les séquences enregistrées à faible fréquence, l'enregistreur de donnée stocke la valeur min, moyenne et max observée durant chaque intervalle de temps dans des colonnes avec les suffixes \_min, \_avg et \_max respectivement.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

une liste de chaîne de caractères.

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

**datastream→get\_data()****YDataStream****datastream→data()datastream.get\_data()**

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

js	function <b>get_data( row, col)</b>
nodejs	function <b>get_data( row, col)</b>
php	function <b>get_data( \$row, \$col)</b>
cpp	<b>double get_data( int row, int col)</b>
m	- <b>(double) data : (int) row</b> <b>: (int) col</b>
pas	<b>function get_data( row: LongInt, col: LongInt): double</b>
vb	<b>function get_data( ) As Double</b>
cs	<b>double get_data( int row, int col)</b>
java	<b>double get_data( int row, int col)</b>
py	<b>def get_data( row, col)</b>

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).

```
js function get_dataRows( )
nodejs function get_dataRows( )
php function get_dataRows( )
cpp vector< vector<double> > get_dataRows( )
m -(NSMutableArray*) dataRows
pas function get_dataRows( ): TDoubleArrayList
vb function get_dataRows( ) As List
cs List<List<double>> get_dataRows( )
java ArrayList<ArrayList<Double>> get_dataRows( )
py def get_dataRows( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

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

**Retourne :**

une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

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

**datastream→get\_dataSamplesIntervalMs()**  
**datastream→dataSamplesIntervalMs()**  
**datastream.get\_dataSamplesIntervalMs()****YDataStream**

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

js	function <b>get_dataSamplesIntervalMs( )</b>
node.js	function <b>get_dataSamplesIntervalMs( )</b>
php	function <b>get_dataSamplesIntervalMs( )</b>
cpp	int <b>get_dataSamplesIntervalMs( )</b>
m	-(int) <b>dataSamplesIntervalMs</b>
pas	function <b>get_dataSamplesIntervalMs( )</b> : LongInt
vb	function <b>get_dataSamplesIntervalMs( )</b> As Integer
cs	int <b>get_dataSamplesIntervalMs( )</b>
java	int <b>get_dataSamplesIntervalMs( )</b>
py	def <b>get_dataSamplesIntervalMs( )</b>

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.

js	function <b>get_duration( )</b>
node.js	function <b>get_duration( )</b>
php	function <b>get_duration( )</b>
cpp	int <b>get_duration( )</b>
m	-(int) duration
pas	function <b>get_duration( ): LongInt</b>
vb	function <b>get_duration( ) As Integer</b>
cs	int <b>get_duration( )</b>
java	int <b>get_duration( )</b>
py	def <b>get_duration( )</b>

**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.

js	function <b>get_maxValue( )</b>
node.js	function <b>get_maxValue( )</b>
php	function <b>get_maxValue( )</b>
cpp	double <b>get_maxValue( )</b>
m	-(double) maxValue
pas	function <b>get_maxValue( )</b> : double
vb	function <b>get_maxValue( )</b> As Double
cs	double <b>get_maxValue( )</b>
java	double <b>get_maxValue( )</b>
py	def <b>get_maxValue( )</b>

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.

```
js function get_minValue( )
node.js function get_minValue( )
php function get_minValue( )
cpp double get_minValue( )
m -(double) minValue
pas function get_minValue( ): double
vb function get_minValue( ) As Double
cs double get_minValue( )
java double get_minValue( )
py def get_minValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus petite valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

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

**datastream→getRowCount()****YDataStream****datastream→rowCount()datastream.getRowCount()**

Retourne le nombre d'enregistrement contenus dans la séquence.

js	function <b>getRowCount( )</b>
nodejs	function <b>getRowCount( )</b>
php	function <b>getRowCount( )</b>
cpp	int <b>getRowCount( )</b>
m	-(int) rowCount
pas	function <b>getRowCount( ): LongInt</b>
vb	function <b>getRowCount( ) As Integer</b>
cs	int <b>getRowCount( )</b>
java	int <b>getRowCount( )</b>
py	def <b>getRowCount( )</b>

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.

```
js function get_runIndex( )  
node.js function get_runIndex( )  
php function get_runIndex( )  
cpp int get_runIndex( )  
m -(int) runIndex  
pas function get_runIndex( ): LongInt  
vb function get_runIndex( ) As Integer  
cs int get_runIndex( )  
java int get_runIndex( )  
py def get_runIndex( )
```

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

**Retourne :**

un entier positif correspondant au numéro du Run

**datastream→getStartTime()****YDataStream****datastream→startTime()datastream.getStartTime()**

Retourne le temps de départ relatif de la séquence (en secondes).

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

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).

```
js function getStartTimeUTC( )
nodejs function getStartTimeUTC( )
php function getStartTimeUTC( )
cpp s64 getStartTimeUTC( )
m -(s64) startTimeUTC
pas function getStartTimeUTC( ): int64
vb function getStartTimeUTC( ) As Long
cs long getStartTimeUTC( )
java long getStartTimeUTC( )
py def 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>
node.js 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.
<b>digitalio→get_functionId()</b>	Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.
<b>digitalio→get_hardwareId()</b>	Retourne l'identifiant matériel unique du port d'E/S digital au format SERIAL . FUNCTIONID.
<b>digitalio→get_logicalName()</b>	Retourne le nom logique du port d'E/S digital.
<b>digitalio→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>digitalio→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>digitalio→get_outputVoltage()</b>	Retourne la source de tension utilisée pour piloter les bits en sortie.
<b>digitalio→get_portDirection()</b>	Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.
<b>digitalio→get_portOpenDrain()</b>	Retourne le type d'interface électrique de chaque bit du port (bitmap).
<b>digitalio→get_portPolarity()</b>	Retourne la polarité des bits du port (bitmap).
<b>digitalio→get_portSize()</b>	Retourne le nombre de bits implémentés dans le port d'E/S.
<b>digitalio→get_portState()</b>	Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.
<b>digitalio→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>digitalio→isOnline()</b>	Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.
<b>digitalio→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.
<b>digitalio→load(msValidity)</b>	Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.
<b>digitalio→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.
<b>digitalio→nextDigitalIO()</b>	Continue l'énumération des ports d'E/S digitaux commencée à l'aide de yFirstDigitalIO( ).
<b>digitalio→pulse(bitno, ms_duration)</b>	Déclenche une impulsion de durée spécifiée sur un bit choisi.
<b>digitalio→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>digitalio→set_bitDirection(bitno, bitdirection)</b>	Change la direction d'un seul bit du port d'E/S.
<b>digitalio→set_bitOpenDrain(bitno, opendrain)</b>	Change le type d'interface électrique d'un seul bit du port d'E/S.
<b>digitalio→set_bitPolarity(bitno, bitpolarity)</b>	Change la polarité d'un seul bit du port d'E/S.

**digitalio→set\_bitState(bitno, bitstate)**

Change l'état d'un seul bit du port d'E/S.

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

Modifie le nom logique du port d'E/S digital.

**digitalio→set\_outputVoltage(newval)**

Modifie la source de tension utilisée pour piloter les bits en sortie.

**digitalio→set\_portDirection(newval)**

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

**digitalio→set\_portOpenDrain(newval)**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

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

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne 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() yFindDigitalIO()YDigitalIO.FindDigitalIO()

**YDigitalIO**

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

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

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

**func** une chaîne de caractères qui référence le port d'E/S digital sans ambiguïté

### Retourne :

un objet de classe `YDigitalIO` qui permet ensuite de contrôler le port d'E/S digital.

## YDigitalIO.FirstDigitalIO()

### yFirstDigitalIO() YDigitalIO.FirstDigitalIO()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

```
js function yFirstDigitalIO( )
nodejs function FirstDigitalIO( )
php function yFirstDigitalIO( )
cpp YDigitalIO* yFirstDigitalIO( )
m YDigitalIO* yFirstDigitalIO( )
pas function yFirstDigitalIO( ): TYDigitalIO
vb function yFirstDigitalIO( ) As YDigitalIO
cs YDigitalIO FirstDigitalIO( )
java YDigitalIO FirstDigitalIO( )
py def FirstDigitalIO( )
```

Utiliser la fonction `YDigitalIO.nextDigitalIO()` pour itérer sur les autres ports d'E/S digitaux.

#### Retourne :

un pointeur sur un objet `YDigitalIO`, correspondant à le 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.

```

js function delayedPulse( bitno, ms_delay, ms_duration)
nodejs function delayedPulse( bitno, ms_delay, ms_duration)
php function delayedPulse( $bitno, $ms_delay, $ms_duration)
cpp int delayedPulse( int bitno, int ms_delay, int ms_duration)
m -(int) delayedPulse : (int) bitno : (int) ms_delay : (int) ms_duration
pas function delayedPulse( bitno: LongInt,
                           ms_delay: LongInt,
                           ms_duration: LongInt): LongInt

vb function delayedPulse( ) As Integer
cs int delayedPulse( int bitno, int ms_delay, int ms_duration)
java int delayedPulse( int bitno, int ms_delay, int ms_duration)
py def delayedPulse( bitno, ms_delay, ms_duration)
cmd YDigitalIO target 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.

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

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

**Retourne :**

une chaîne de caractères décrivant le port d'E/S digital (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

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

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

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

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

(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.

js	function <b>get_bitOpenDrain( bitno)</b>
node.js	function <b>get_bitOpenDrain( bitno)</b>
php	function <b>get_bitOpenDrain( \$bitno)</b>
cpp	int <b>get_bitOpenDrain( int bitno)</b>
m	- (int) <b>bitOpenDrain : (int) bitno</b>
pas	function <b>get_bitOpenDrain( bitno: LongInt): LongInt</b>
vb	function <b>get_bitOpenDrain( ) As Integer</b>
cs	int <b>get_bitOpenDrain( int bitno)</b>
java	int <b>get_bitOpenDrain( int bitno)</b>
py	def <b>get_bitOpenDrain( bitno)</b>
cmd	<b>YDigitalIO target get_bitOpenDrain bitno</b>

**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.

js	function <b>get_bitPolarity( bitno)</b>
node.js	function <b>get_bitPolarity( bitno)</b>
php	function <b>get_bitPolarity( \$bitno)</b>
cpp	int <b>get_bitPolarity( int bitno)</b>
m	-(int) <b>bitPolarity : (int) bitno</b>
pas	function <b>get_bitPolarity( bitno: LongInt): LongInt</b>
vb	function <b>get_bitPolarity( ) As Integer</b>
cs	int <b>get_bitPolarity( int bitno)</b>
java	int <b>get_bitPolarity( int bitno)</b>
py	def <b>get_bitPolarity( bitno)</b>
cmd	<b>YDigitalIO target get_bitPolarity bitno</b>

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()**  
**digitalio→bitState()digitalio.get\_bitState()****YDigitalIO**

Retourne l'état d'un seul bit du port d'E/S.

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

**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.

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

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.

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

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.

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

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.

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

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

**Retourne :**

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

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

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

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

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.

```
js function get.hardwareId( )  
node.js function get.hardwareId( )  
php function get.hardwareId( )  
cpp string get.hardwareId( )  
m -(NSString*) hardwareId  
vb function get.hardwareId( ) As String  
cs string get.hardwareId( )  
java String get.hardwareId( )  
py def get.hardwareId( )
```

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.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YDigitalIO target 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()**  
**digitalio→module()digitalio.get\_module()****YDigitalIO**

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

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

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()

YDigitalIO

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

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

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

Retourne la source de tension utilisée pour piloter les bits en sortie.

```
js function get_outputVoltage( )  
nodejs function get_outputVoltage( )  
php function get_outputVoltage( )  
cpp Y_OUTPUTVOLTAGE_enum get_outputVoltage( )  
m -(Y_OUTPUTVOLTAGE_enum) outputVoltage  
pas function get_outputVoltage( ): Integer  
vb function get_outputVoltage( ) As Integer  
cs int get_outputVoltage( )  
java int get_outputVoltage( )  
py def get_outputVoltage( )  
cmd YDigitalIO target 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.

js	function <b>get_portDirection( )</b>
nodejs	function <b>get_portDirection( )</b>
php	function <b>get_portDirection( )</b>
cpp	<b>int get_portDirection( )</b>
m	- <b>(int) portDirection</b>
pas	function <b>get_portDirection( )</b> : LongInt
vb	function <b>get_portDirection( )</b> As Integer
cs	<b>int get_portDirection( )</b>
java	<b>int get_portDirection( )</b>
py	<b>def get_portDirection( )</b>
cmd	<b>YDigitalIO target get_portDirection</b>

**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).

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

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

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.

js	function <b>get_portSize( )</b>
node.js	function <b>get_portSize( )</b>
php	function <b>get_portSize( )</b>
cpp	int <b>get_portSize( )</b>
m	-(int) <b>portSize</b>
pas	function <b>get_portSize( ): LongInt</b>
vb	function <b>get_portSize( ) As Integer</b>
cs	int <b>get_portSize( )</b>
java	int <b>get_portSize( )</b>
py	def <b>get_portSize( )</b>
cmd	<b>YDigitalIO target get_portSize</b>

**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.

js	function <b>get_portState( )</b>
nodejs	function <b>get_portState( )</b>
php	function <b>get_portState( )</b>
cpp	int <b>get_portState( )</b>
m	-(int) portState
pas	function <b>get_portState( ): LongInt</b>
vb	function <b>get_portState( ) As Integer</b>
cs	int <b>get_portState( )</b>
java	int <b>get_portState( )</b>
py	def <b>get_portState( )</b>
cmd	<b>YDigitalIO target get_portState</b>

**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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData)  
m -(void*) userData  
pas function get(userData): Object  
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.

## digitalio→isOnline()digitalio.isOnline()

## YDigitalIO

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()

YDigitalIO

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

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

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()

YDigitalIO

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

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

**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.

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

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

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

### Paramètres :

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

**digitalio→set\_bitDirection()**

YDigitalIO

**digitalio→setBitDirection()digitalio.set\_bitDirection()**

Change la direction d'un seul bit du port d'E/S.

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

**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.

js	function <b>set_bitOpenDrain( bitno, opendrain)</b>
node.js	function <b>set_bitOpenDrain( bitno, opendrain)</b>
php	function <b>set_bitOpenDrain( \$bitno, \$opendrain)</b>
cpp	int <b>set_bitOpenDrain( int bitno, int opendrain)</b>
m	- <b>(int) setBitOpenDrain : (int) bitno : (int) opendrain</b>
pas	function <b>set_bitOpenDrain( bitno: LongInt, opendrain: LongInt): LongInt</b>
vb	function <b>set_bitOpenDrain( ) As Integer</b>
cs	int <b>set_bitOpenDrain( int bitno, int opendrain)</b>
java	int <b>set_bitOpenDrain( int bitno, int opendrain)</b>
py	def <b>set_bitOpenDrain( bitno, opendrain)</b>
cmd	YDigitalIO <b>target set_bitOpenDrain bitno opendrain</b>

#### 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.

<b>js</b>	<code>function set_bitPolarity( bitno, bitpolarity)</code>
<b>node.js</b>	<code>function set_bitPolarity( bitno, bitpolarity)</code>
<b>php</b>	<code>function set_bitPolarity( \$bitno, \$bitpolarity)</code>
<b>cpp</b>	<code>int set_bitPolarity( int bitno, int bitpolarity)</code>
<b>m</b>	<code>-(int) setBitPolarity : (int) bitno : (int) bitpolarity</code>
<b>pas</b>	<code>function set_bitPolarity( bitno: LongInt, bitpolarity: LongInt): LongInt</code>
<b>vb</b>	<code>function set_bitPolarity( ) As Integer</code>
<b>cs</b>	<code>int set_bitPolarity( int bitno, int bitpolarity)</code>
<b>java</b>	<code>int set_bitPolarity( int bitno, int bitpolarity)</code>
<b>py</b>	<code>def set_bitPolarity( bitno, bitpolarity)</code>
<b>cmd</b>	<code>YDigitalIO target set_bitPolarity bitno bitpolarity</code>

**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.

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

**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.

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

js	function <b>set_outputVoltage( newval)</b>
node.js	function <b>set_outputVoltage( newval)</b>
php	function <b>set_outputVoltage( \$newval)</b>
cpp	int <b>set_outputVoltage( Y_OUTPUTVOLTAGE_enum newval)</b>
m	-(int) <b>setOutputVoltage : (Y_OUTPUTVOLTAGE_enum) newval</b>
pas	function <b>set_outputVoltage( newval: Integer): integer</b>
vb	function <b>set_outputVoltage( ByVal newval As Integer) As Integer</b>
cs	int <b>set_outputVoltage( int newval)</b>
java	int <b>set_outputVoltage( int newval)</b>
py	def <b>set_outputVoltage( newval)</b>
cmd	<b>YDigitalIO target set_outputVoltage newval</b>

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.

js	function set_portDirection( newval)
nodejs	function set_portDirection( newval)
php	function set_portDirection( \$newval)
cpp	int set_portDirection( int newval)
m	-(int) setPortDirection : (int) newval
pas	function set_portDirection( newval: LongInt): integer
vb	function set_portDirection( ByVal newval As Integer) As Integer
cs	int set_portDirection( int newval)
java	int set_portDirection( int newval)
py	def set_portDirection( newval)
cmd	YDigitalIO target 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).

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

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.

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

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

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).

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

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.

js	function <b>toggle_bitState( bitno)</b>
node.js	function <b>toggle_bitState( bitno)</b>
php	function <b>toggle_bitState( \$bitno)</b>
cpp	int <b>toggle_bitState( int bitno)</b>
m	-( <b>int</b> ) <b>toggle_bitState : (int) bitno</b>
pas	function <b>toggle_bitState( bitno: LongInt): LongInt</b>
vb	function <b>toggle_bitState( ) As Integer</b>
cs	int <b>toggle_bitState( int bitno)</b>
java	int <b>toggle_bitState( int bitno)</b>
py	def <b>toggle_bitState( bitno)</b>
cmd	YDigitalIO <b>target toggle_bitState bitno</b>

**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()

YDigitalIO

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

```
js function wait_async( callback, context)
node.js 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>
node.js 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 *

```

### Fonction globales

#### yFindDisplay(func)

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

#### yFirstDisplay()

Commence l'énumération des écrans 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()

### 3. Reference

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()YDisplay.FindDisplay()

**YDisplay**

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

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

L'identifiant peut être spécifié sous plusieurs formes:

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

Cette fonction n'exige pas que 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() YDisplay.FirstDisplay()

## YDisplay

Commence l'énumération des écran accessibles par la librairie.

js	function <b>yFirstDisplay( )</b>
nodejs	function <b>FirstDisplay( )</b>
php	function <b>yFirstDisplay( )</b>
cpp	YDisplay* <b>yFirstDisplay( )</b>
m	YDisplay* <b>yFirstDisplay( )</b>
pas	function <b>yFirstDisplay( ): TYDisplay</b>
vb	function <b>yFirstDisplay( ) As YDisplay</b>
cs	YDisplay <b>FirstDisplay( )</b>
java	YDisplay <b>FirstDisplay( )</b>
py	def <b>FirstDisplay( )</b>

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écran.

#### Retourne :

un pointeur sur un objet `YDisplay`, correspondant à le 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.

```

js   function copyLayerContent( srcLayerId, dstLayerId)
node.js function copyLayerContent( srcLayerId, dstLayerId)
php  function copyLayerContent( $srcLayerId, $dstLayerId)
cpp   int copyLayerContent( int srcLayerId, int dstLayerId)
m     -(int) copyLayerContent : (int) srcLayerId
                  : (int) dstLayerId
pas   function copyLayerContent( srcLayerId: LongInt,
                               dstLayerId: LongInt): LongInt
vb    function copyLayerContent( ) As Integer
cs    int copyLayerContent( int srcLayerId, int dstLayerId)
java  int copyLayerContent( int srcLayerId, int dstLayerId)
py    def copyLayerContent( srcLayerId, dstLayerId)
cmd   YDisplay target copyLayerContent srcLayerId dstLayerId

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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'écran (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

## display→fade()`display.fade()`

YDisplay

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

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

### 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	<b>YDisplay target get_advertisedValue</b>

**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).

```
js function get_brightness( )  
node.js function get_brightness( )  
php function get_brightness( )  
cpp int get_brightness( )  
m -(int) brightness  
pas function get_brightness( ): LongInt  
vb function get_brightness( ) As Integer  
cs int get_brightness( )  
java int get_brightness( )  
py def get_brightness( )  
cmd YDisplay target 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.

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

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

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

## display→get\_displayLayer() display→displayLayer()display.get\_displayLayer()

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

```
js function get_displayLayer( layerId)
node.js function get_displayLayer( layerId)
php function get_displayLayer( $layerId)
cpp YDisplayLayer* get_displayLayer( unsigned layerId)
m -(YDisplayLayer*) displayLayer : (unsigned) layerId
vb function get_displayLayer( ) As YDisplayLayer
cs YDisplayLayer get_displayLayer( int layerId)
java synchronized YDisplayLayer get_displayLayer( int layerId)
py def get_displayLayer( layerId)
```

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

### Paramètres :

**layerId** l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

### Retourne :

un objet YDisplayLayer

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

## display→get\_displayType() display→displayType()display.get\_displayType()

YDisplay

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

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

### 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.

```
js function get_displayWidth( )  
node.js function get_displayWidth( )  
php function get_displayWidth( )  
cpp int get_displayWidth( )  
m -(int) displayWidth  
pas function get_displayWidth( ): LongInt  
vb function get_displayWidth( ) As Integer  
cs int get_displayWidth( )  
java int get_displayWidth( )  
py def get_displayWidth( )  
cmd YDisplay target 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() display→enabled()display.get\_enabled()

YDisplay

Retourne vrai si le l'écran est alimenté, faux sinon.

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

### 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→get\_errorMessage()****YDisplay****display→errorMessage()display.getErrorMessage()**

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

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

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.

js	function <b>get_errorType( )</b>
nodejs	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	YRETCODE <b>get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	YRETCODE <b>get_errorType( )</b>
java	int <b>get_errorType( )</b>
py	def <b>get_errorType( )</b>

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.

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

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.

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

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

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

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

**display→get\_functionId()  
display→functionId()display.get\_functionId()****YDisplay**

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

```
js function get_functionId( )
node.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( )
py def 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()**  
**display→hardwareId()display.get\_hardwareId()****YDisplay**

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

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

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

**Retourne :**

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

**display→get\_layerCount()** **YDisplay**  
**display→layerCount()display.get\_layerCount()**

Retourne le nombre des couches affichables disponibles.

```
js function get_layerCount( )
node.js function get_layerCount( )
php function get_layerCount( )
cpp int get_layerCount( )
m -(int) layerCount
pas function get_layerCount( ): LongInt
vb function get_layerCount( ) As Integer
cs int get_layerCount( )
java int get_layerCount( )
py def get_layerCount( )
cmd YDisplay target get_layerCount
```

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

YDisplay

Retourne la hauteur des couches affichables, en pixels.

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

**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.

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

**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()**  
**display→logicalName()display.get\_logicalName()****YDisplay**

Retourne le nom logique de l'écran.

js	function <b>get_logicalName( )</b>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	<b>YDisplay target get_logicalName</b>

**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.

```
js function get_module( )
node.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`

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

YDisplay

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

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

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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() display→orientation()display.get\_orientation()

YDisplay

Retourne l'orientation sélectionnée pour l'écran.

```
js function get_orientation( )
node.js function get_orientation( )
php function get_orientation( )
cpp Y_ORIENTATION_enum get_orientation( )
m -(Y_ORIENTATION_enum) orientation
pas function get_orientation( ): Integer
vb function get_orientation( ) As Integer
cs int get_orientation( )
java int get_orientation( )
py def get_orientation( )
cmd YDisplay target 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() display→startupSeq()display.get\_startupSeq()

YDisplay

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

```
js function get_startupSeq( )  
nodejs function get_startupSeq( )  
php function get_startupSeq( )  
cpp string get_startupSeq( )  
m -(NSString*) startupSeq  
pas function get_startupSeq( ): string  
vb function get_startupSeq( ) As String  
cs string get_startupSeq( )  
java String get_startupSeq( )  
py def get_startupSeq( )  
cmd YDisplay target 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) display→userData()display.get(userData)

YDisplay

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

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData) {  
m -(void*) userData  
pas function get(userData): Object  
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.

**display→isOnline()display.isOnline()****YDisplay**

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

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	<b>def isOnline( )</b>

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()

YDisplay

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()

YDisplay

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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.

```
js function newSequence( )
nodejs function newSequence()
php function newSequence()
cpp int newSequence( )
m -(int) newSequence
pas function newSequence( ): LongInt
vb function newSequence( ) As Integer
cs int newSequence( )
java int newSequence( )
py def newSequence( )
cmd YDisplay target newSequence
```

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()`.

js	function <b>nextDisplay()</b>
nodejs	function <b>nextDisplay()</b>
php	function <b>nextDisplay()</b>
cpp	<b>YDisplay * nextDisplay()</b>
m	<b>-(YDisplay*) nextDisplay</b>
pas	function <b>nextDisplay()</b> : TYDisplay
vb	function <b>nextDisplay()</b> As YDisplay
cs	<b>YDisplay nextDisplay()</b>
java	<b>YDisplay nextDisplay()</b>
py	<b>def nextDisplay()</b>

**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.

```

js   function pauseSequence( delay_ms)
nodejs function pauseSequence( delay_ms)
php  function pauseSequence( $delay_ms)
cpp   int pauseSequence( int delay_ms)
m    -(int) pauseSequence : (int) delay_ms
pas   function pauseSequence( delay_ms: LongInt): LongInt
vb    function pauseSequence( ) As Integer
cs   int pauseSequence( int delay_ms)
java  int pauseSequence( int delay_ms)
py    def pauseSequence( delay_ms)
cmd   YDisplay target pauseSequence delay_ms

```

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( ).

```
js function playSequence( sequenceName)
nodejs function playSequence( sequenceName)
php function playSequence( $sequenceName)
cpp int playSequence( string sequenceName)
m -(int) playSequence : (NSString*) sequenceName
pas function playSequence( sequenceName: string): LongInt
vb function playSequence( ) As Integer
cs int playSequence( string sequenceName)
java int playSequence( String sequenceName)
py def playSequence( sequenceName)
cmd YDisplay target playSequence sequenceName
```

**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.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
node.js	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YDisplayValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YDisplayValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYDisplayValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**display→resetAll()display.resetAll()****YDisplay**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

```
js function resetAll( )
nodejs function resetAll( )
php function resetAll( )
cpp int resetAll( )
m -(int) resetAll
pas function resetAll( ): LongInt
vb function resetAll( ) As Integer
cs int resetAll( )
java int resetAll( )
py def resetAll( )
cmd YDisplay target 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.

js	function saveSequence( sequenceName)
nodejs	function saveSequence( sequenceName)
php	function saveSequence( \$sequenceName)
cpp	int saveSequence( string sequenceName)
m	-(int) saveSequence : (NSString*) sequenceName
pas	function saveSequence( sequenceName: string): LongInt
vb	function saveSequence( ) As Integer
cs	int saveSequence( string sequenceName)
java	int saveSequence( String sequenceName)
py	def saveSequence( sequenceName)
cmd	YDisplay target saveSequence sequenceName

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_brightness()**

YDisplay

**display→setBrightness()display.set\_brightness()**

Modifie la luminositéde l'écran.

```
js function set_brightness( newval)
node.js function set_brightness( newval)
php function set_brightness( $newval)
cpp int set_brightness( int newval)
m -(int) setBrightness : (int) newval
pas function set_brightness( newval: LongInt): integer
vb function set_brightness( ByVal newval As Integer) As Integer
cs int set_brightness( int newval)
java int set_brightness( int newval)
py def set_brightness( newval)
cmd YDisplay target 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() display→setEnabled()display.set\_enabled()

YDisplay

Modifie l'état d'activité de l'écran.

```
js function set_enabled( newval)
nodejs function set_enabled( newval)
php function set_enabled( $newval)
cpp int set_enabled( Y_ENABLED_enum newval)
m -(int) setEnabled : (Y_ENABLED_enum) newval
pas function set_enabled( newval: Integer): integer
vb function set_enabled( ByVal newval As Integer) As Integer
cs int set_enabled( int newval)
java int set_enabled( int newval)
py def set_enabled( newval)
cmd YDisplay target set_enabled newval
```

### 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() display→setLogicalName()display.set\_logicalName()

YDisplay

Modifie le nom logique de l'écran.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YDisplay 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 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.

js	function <b>set_orientation( newval)</b>
node.js	function <b>set_orientation( newval)</b>
php	function <b>set_orientation( \$newval)</b>
cpp	int <b>set_orientation( Y_ORIENTATION_enum newval)</b>
m	-(int) setOrientation : (Y_ORIENTATION_enum) <b>newval</b>
pas	function <b>set_orientation( newval: Integer): integer</b>
vb	function <b>set_orientation( ByVal newval As Integer) As Integer</b>
cs	int <b>set_orientation( int newval)</b>
java	int <b>set_orientation( int newval)</b>
py	def <b>set_orientation( newval)</b>
cmd	<b>YDisplay target set_orientation newval</b>

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() display→setStartupSeq()display.set\_startupSeq()

YDisplay

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
js function set_startupSeq( newval)
node.js function set_startupSeq( newval)
php function set_startupSeq( $newval)
cpp int set_startupSeq( const string& newval)
m -(int) setStartupSeq : (NSString*) newval
pas function set_startupSeq( newval: string): integer
vb function set_startupSeq( ByVal newval As String) As Integer
cs int set_startupSeq( string newval)
java int set_startupSeq( String newval)
py def set_startupSeq( newval)
cmd YDisplay target 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) display→setUserData()display.set(userData)

YDisplay

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
js function setUserData( data)
nodejs function setUserData( data)
php function setUserData( $data)
cpp void setUserData( void* data)
m -(void) setUserData : (void*) data
pas procedure setUserData( data: Tobject)
vb procedure setUserData( ByVal data As Object)
cs void setUserData( object data)
java void setUserData( Object data)
py def setUserData( data)
```

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.

```
js function stopSequence( )  
nodejs function stopSequence( )  
php function stopSequence( )  
cpp int stopSequence( )  
m -(int) stopSequence  
pas function stopSequence( ): LongInt  
vb function stopSequence( ) As Integer  
cs int stopSequence( )  
java int stopSequence( )  
py def stopSequence( )  
cmd YDisplay target stopSequence
```

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.

```

js   function swapLayerContent( layerIdA, layerIdB)
nodejs function swapLayerContent( layerIdA, layerIdB)
php  function swapLayerContent( $layerIdA, $layerIdB)
cpp   int swapLayerContent( int layerIdA, int layerIdB)
m     -(int) swapLayerContent : (int) layerIdA
          : (int) layerIdB

pas  function swapLayerContent( layerIdA: LongInt, layerIdB: LongInt): LongInt
vb   function swapLayerContent( ) As Integer
cs   int swapLayerContent( int layerIdA, int layerIdB)
java int swapLayerContent( int layerIdA, int layerIdB)
py   def swapLayerContent( layerIdA, layerIdB)
cmd  YDisplay target swapLayerContent layerIdA layerIdB

```

La couleur et la transparence de tous les pixels des deux couches sont permutées. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**layerIdA** l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

**layerIdB** l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## display→upload()**display.upload()**

YDisplay

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

```
js function upload( pathname, content)
nodejs function upload( pathname, content)
php function upload( $pathname, $content)
cpp int upload( string pathname, string content)
m -(int) upload : (NSString*) pathname
               : (NSData*) content
pas function upload( pathname: string, content: TByteArray): LongInt
vb procedure upload()
cs int upload( string pathname)
java int upload( String pathname)
py def upload( pathname, content)
cmd YDisplay target 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()****YDisplay**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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.

**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.

js	function <b>clear( )</b>
nodejs	function <b>clear( )</b>
php	function <b>clear( )</b>
cpp	int <b>clear( )</b>
m	-(int) <b>clear</b>
pas	function <b>clear( )</b> : LongInt
vb	function <b>clear( )</b> As Integer
cs	int <b>clear( )</b>
java	int <b>clear( )</b>
py	def <b>clear( )</b>
cmd	<b>YDisplay target [-layer layerId] clear</b>

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.

js	function clearConsole( )
nodejs	function clearConsole( )
php	function clearConsole( )
cpp	int clearConsole( )
m	-(int) clearConsole
pas	function clearConsole( ): LongInt
vb	function clearConsole( ) As Integer
cs	int clearConsole( )
java	int clearConsole( )
py	def clearConsole( )
cmd	YDisplay target [-layer layerId] 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.

```
js function consoleOut( text)
nodejs function consoleOut( text)
php function consoleOut( $text)
cpp int consoleOut( string text)
m -(int) consoleOut : (NSString*) text
pas function consoleOut( text: string): LongInt
vb function consoleOut( ) As Integer
cs int consoleOut( string text)
java int consoleOut( String text)
py def consoleOut( text)
cmd YDisplay target [-layer layerId] 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.

```

js   function drawBar( x1, y1, x2, y2)
node.js function drawBar( x1, y1, x2, y2)
php  function drawBar( $x1, $y1, $x2, $y2)
cpp   int drawBar( int x1, int y1, int x2, int y2)
m    -(int) drawBar : (int) x1
      : (int) y1
      : (int) x2
      : (int) y2
pas   function drawBar( x1: LongInt,
                      y1: LongInt,
                      x2: LongInt,
                      y2: LongInt): LongInt
vb    function drawBar( ) As Integer
cs    int drawBar( int x1, int y1, int x2, int y2)
java   int drawBar( int x1, int y1, int x2, int y2)
py    def drawBar( x1, y1, x2, y2)
cmd   YDisplay target [-layer layerId] drawBar x1 y1 x2 y2

```

**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.

```

js function drawBitmap( x, y, w, bitmap, bgcol)
node.js function drawBitmap( x, y, w, bitmap, bgcol)
php function drawBitmap( $x, $y, $w, $bitmap, $bgcol)
cpp int drawBitmap( int x, int y, int w, string bitmap, int bgcol)
m -(int) drawBitmap : (int) x
           : (int) y
           : (int) w
           : (NSData*) bitmap
           : (int) bgcol

pas function drawBitmap( x: LongInt,
                        y: LongInt,
                        w: LongInt,
                        bitmap: TByteArray,
                        bgcol: LongInt): LongInt

vb procedure drawBitmap( )
cs int drawBitmap( int x, int y, int w, int bgcol)
java int drawBitmap( int x, int y, int w, int bgcol)
py def drawBitmap( x, y, w, bitmap, bgcol)
cmd YDisplay target [-layer layerId] drawBitmap x y w bitmap bgcol

```

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.

<b>js</b>	function <b>drawCircle( x, y, r)</b>
<b>node.js</b>	function <b>drawCircle( x, y, r)</b>
<b>php</b>	function <b>drawCircle( \$x, \$y, \$r)</b>
<b>cpp</b>	int <b>drawCircle( int x, int y, int r)</b>
<b>m</b>	- <b>(int) drawCircle : (int) x</b> <b>: (int) y</b> <b>: (int) r</b>
<b>pas</b>	function <b>drawCircle( x: LongInt, y: LongInt, r: LongInt): LongInt</b>
<b>vb</b>	function <b>drawCircle( ) As Integer</b>
<b>cs</b>	int <b>drawCircle( int x, int y, int r)</b>
<b>java</b>	int <b>drawCircle( int x, int y, int r)</b>
<b>py</b>	def <b>drawCircle( x, y, r)</b>
<b>cmd</b>	<b>YDisplay target [-layer layerId] drawCircle x y r</b>

**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.

```
js function drawDisc( x, y, r)
nodejs function drawDisc( x, y, r)
php function drawDisc( $x, $y, $r)
cpp int drawDisc( int x, int y, int r)
m -(int) drawDisc : (int) x
           : (int) y
           : (int) r
pas function drawDisc( x: LongInt, y: LongInt, r: LongInt): LongInt
vb function drawDisc( ) As Integer
cs int drawDisc( int x, int y, int r)
java int drawDisc( int x, int y, int r)
py def drawDisc( x, y, r)
cmd YDisplay target [-layer layerId] 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.

```

js   function drawImage( x, y, imagename)
node.js function drawImage( x, y, imagename)
php  function drawImage( $x, $y, $imagename)
cpp   int drawImage( int x, int y, string imagename)
m     -(int) drawImage : (int) x
          : (int) y
          : (NSString*) imagename

pas  function drawImage( x: LongInt, y: LongInt, imagename: string): LongInt
vb   function drawImage( ) As Integer
cs   int drawImage( int x, int y, string imagename)
java int drawImage( int x, int y, String imagename)
py   def drawImage( x, y, imagename)
cmd  YDisplay target [-layer layerId] drawImage x y imagename

```

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.

```
js function drawPixel( x, y)
nodejs function drawPixel( x, y)
php function drawPixel( $x, $y)
cpp int drawPixel( int x, int y)
m -(int) drawPixel : (int) x
: (int) y
pas function drawPixel( x: LongInt, y: LongInt): LongInt
vb function drawPixel( ) As Integer
cs int drawPixel( int x, int y)
java int drawPixel( int x, int y)
py def drawPixel( x, y)
cmd YDisplay target [-layer layerId] drawPixel x y
```

**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.

```

js   function drawRect( x1, y1, x2, y2)
node.js function drawRect( x1, y1, x2, y2)
php  function drawRect( $x1, $y1, $x2, $y2)
cpp   int drawRect( int x1, int y1, int x2, int y2)
m    -(int) drawRect : (int) x1
      : (int) y1
      : (int) x2
      : (int) y2
pas   function drawRect( x1: LongInt,
                      y1: LongInt,
                      x2: LongInt,
                      y2: LongInt): LongInt
vb    function drawRect( ) As Integer
cs    int drawRect( int x1, int y1, int x2, int y2)
java  int drawRect( int x1, int y1, int x2, int y2)
py    def drawRect( x1, y1, x2, y2)
cmd   YDisplay target [-layer layerId] 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.

```

js function drawText( x, y, anchor, text)
nodejs function drawText( x, y, anchor, text)
php function drawText( $x, $y, $anchor, $text)
cpp int drawText( int x, int y, Y_ALIGN anchor, string text)
m -(int) drawText : (int) x
           : (int) y
           : (Y_ALIGN) anchor
           : (NSString*) text
pas function drawText( x: LongInt,
                      y: LongInt,
                      anchor: TYALIGN,
                      text: string): LongInt
vb function drawText( ) As Integer
cs int drawText( int x, int y, ALIGN anchor, string text)
java int drawText( int x, int y, ALIGN anchor, String text)
py def drawText( x, y, anchor, text)
cmd YDisplay target [-layer layerId] drawText x y anchor text

```

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.

```
js function get_display( )
nodejs function get_display( )
php function get_display( )
cpp YDisplay* get_display( )
m -(YDisplay*) display
pas function get_display( ): TYDisplay
vb function get_display( ) As YDisplay
cs YDisplay get_display( )
java YDisplay get_display( )
py def get_display( )
```

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.

```
js function get_displayHeight( )  
nodejs function get_displayHeight( )  
php function get_displayHeight( )  
cpp int get_displayHeight( )  
m -(int) displayHeight  
pas function get_displayHeight( ): LongInt  
vb function get_displayHeight( ) As Integer  
cs int get_displayHeight( )  
java int get_displayHeight( )  
py def get_displayHeight( )  
cmd YDisplay target [-layer layerId] 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.

js	function <b>get_displayWidth( )</b>
node.js	function <b>get_displayWidth( )</b>
php	function <b>get_displayWidth( )</b>
cpp	int <b>get_displayWidth( )</b>
m	-(int) <b>displayWidth</b>
pas	function <b>get_displayWidth( )</b> : LongInt
vb	function <b>get_displayWidth( )</b> As Integer
cs	int <b>get_displayWidth( )</b>
java	int <b>get_displayWidth( )</b>
py	def <b>get_displayWidth( )</b>
cmd	YDisplay <b>target [-layer layerId] get_displayWidth</b>

**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.

```
js function get_layerHeight( )  
nodejs function get_layerHeight( )  
php function get_layerHeight( )  
cpp int get_layerHeight( )  
m -(int) layerHeight  
pas function get_layerHeight( ): LongInt  
vb function get_layerHeight( ) As Integer  
cs int get_layerHeight( )  
java int get_layerHeight( )  
py def get_layerHeight( )  
cmd YDisplay target [-layer layerId] 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.

js	function <b>get_layerWidth( )</b>
node.js	function <b>get_layerWidth( )</b>
php	function <b>get_layerWidth( )</b>
cpp	int <b>get_layerWidth( )</b>
m	-(int) <b>layerWidth</b>
pas	function <b>get_layerWidth( )</b> : LongInt
vb	function <b>get_layerWidth( )</b> As Integer
cs	int <b>get_layerWidth( )</b>
java	int <b>get_layerWidth( )</b>
py	def <b>get_layerWidth( )</b>
cmd	<b>YDisplay target [-layer layerId] get_layerWidth</b>

**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.

```
js function hide( )
nodejs function hide( )
php function hide( )
cpp int hide( )
m -(int) hide
pas function hide( ): LongInt
vb function hide( ) As Integer
cs int hide( )
java int hide( )
py def hide( )
cmd YDisplay target [-layer layerId] hide
```

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.

js	function <b>lineTo( x, y)</b>
node.js	function <b>lineTo( x, y)</b>
php	function <b>lineTo( \$x, \$y)</b>
cpp	int <b>lineTo( int x, int y)</b>
m	- <b>(int) lineTo : (int) x</b> <b>: (int) y</b>
pas	function <b>lineTo( x: LongInt, y: LongInt): LongInt</b>
vb	function <b>lineTo( ) As Integer</b>
cs	int <b>lineTo( int x, int y)</b>
java	int <b>lineTo( int x, int y)</b>
py	def <b>lineTo( x, y)</b>
cmd	<b>YDisplay target [-layer layerId] lineTo x y</b>

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.

```
js function moveTo( x, y)
nodejs function moveTo( x, y)
php function moveTo( $x, $y)
cpp int moveTo( int x, int y)
m -(int) moveTo : (int) x
               : (int) y
pas function moveTo( x: LongInt, y: LongInt): LongInt
vb function moveTo( ) As Integer
cs int moveTo( int x, int y)
java int moveTo( int x, int y)
py def moveTo( x, y)
cmd YDisplay target [-layer layerId] 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).

```
js function reset( )
node.js function reset()
php function reset()
cpp int reset( )
m -(int) reset
pas function reset( ): LongInt
vb function reset( ) As Integer
cs int reset( )
java int reset( )
py def reset( )
cmd YDisplay target [-layer layerId] reset
```

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.

```
js function selectColorPen( color)
node.js function selectColorPen( color)
php function selectColorPen( $color)
cpp int selectColorPen( int color)
m -(int) selectColorPen : (int) color
pas function selectColorPen( color: LongInt): LongInt
vb function selectColorPen( ) As Integer
cs int selectColorPen( int color)
java int selectColorPen( int color)
py def selectColorPen( color)
cmd YDisplay target [-layer layerId] selectColorPen color
```

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.

```
js function selectEraser( )
nodejs function selectEraser( )
php function selectEraser( )
cpp int selectEraser( )
m -(int) selectEraser
pas function selectEraser( ): LongInt
vb function selectEraser( ) As Integer
cs int selectEraser( )
java int selectEraser( )
py def selectEraser( )
cmd YDisplay target [-layer layerId] selectEraser
```

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.

```
js function selectFont( fontname)
nodejs function selectFont( fontname)
php function selectFont( $fontname)
cpp int selectFont( string fontname)
m -(int) selectFont : (NSString*) fontname
pas function selectFont( fontname: string): LongInt
vb function selectFont( ) As Integer
cs int selectFont( string fontname)
java int selectFont( String fontname)
py def selectFont( fontname)
cmd YDisplay target [-layer layerId] 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.

js	function <b>selectGrayPen( graylevel)</b>
node.js	function <b>selectGrayPen( graylevel)</b>
php	function <b>selectGrayPen( \$graylevel)</b>
cpp	int <b>selectGrayPen( int graylevel)</b>
m	<b>-(int) selectGrayPen : (int) graylevel</b>
pas	function <b>selectGrayPen( graylevel: LongInt): LongInt</b>
vb	function <b>selectGrayPen( ) As Integer</b>
cs	int <b>selectGrayPen( int graylevel)</b>
java	int <b>selectGrayPen( int graylevel)</b>
py	def <b>selectGrayPen( graylevel)</b>
cmd	<b>YDisplay target [-layer layerId] selectGrayPen graylevel</b>

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.

```
js function setAntialiasingMode( mode)
node.js function setAntialiasingMode( mode)
php function setAntialiasingMode( $mode)
cpp int setAntialiasingMode( bool mode)
m -(int) setAntialiasingMode : (bool) mode
pas function setAntialiasingMode( mode: boolean): LongInt
vb function setAntialiasingMode( ) As Integer
cs int setAntialiasingMode( bool mode)
java int setAntialiasingMode( boolean mode)
py def setAntialiasingMode( mode)
cmd YDisplay target [-layer layerId] 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.

<code>js</code>	<code>function setConsoleBackground( bgcol)</code>
<code>node.js</code>	<code>function setConsoleBackground( bgcol)</code>
<code>php</code>	<code>function setConsoleBackground( \$bgcol)</code>
<code>cpp</code>	<code>int setConsoleBackground( int bgcol)</code>
<code>m</code>	<code>-(int) setConsoleBackground : (int) bgcol</code>
<code>pas</code>	<code>function setConsoleBackground( bgcol: LongInt): LongInt</code>
<code>vb</code>	<code>function setConsoleBackground( ) As Integer</code>
<code>cs</code>	<code>int setConsoleBackground( int bgcol)</code>
<code>java</code>	<code>int setConsoleBackground( int bgcol)</code>
<code>py</code>	<code>def setConsoleBackground( bgcol)</code>
<code>cmd</code>	<code>YDisplay target [-layer layerId] setConsoleBackground bgcol</code>

### 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.

```

js   function setConsoleMargins( x1, y1, x2, y2)
node.js function setConsoleMargins( x1, y1, x2, y2)
php  function setConsoleMargins( $x1, $y1, $x2, $y2)
cpp   int setConsoleMargins( int x1, int y1, int x2, int y2)
m    -(int) setConsoleMargins : (int) x1
           : (int) y1
           : (int) x2
           : (int) y2
pas   function setConsoleMargins( x1: LongInt,
                                  y1: LongInt,
                                  x2: LongInt,
                                  y2: LongInt): LongInt
vb    function setConsoleMargins( ) As Integer
cs    int setConsoleMargins( int x1, int y1, int x2, int y2)
java   int setConsoleMargins( int x1, int y1, int x2, int y2)
py    def setConsoleMargins( x1, y1, x2, y2)
cmd   YDisplay target [-layer layerId] 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.

```
js function setConsoleWordWrap( wordwrap)
nodejs function setConsoleWordWrap( wordwrap)
php function setConsoleWordWrap( $wordwrap)
cpp int setConsoleWordWrap( bool wordwrap)
m -(int) setConsoleWordWrap : (bool) wordwrap
pas function setConsoleWordWrap( wordwrap: boolean): LongInt
vb function setConsoleWordWrap( ) As Integer
cs int setConsoleWordWrap( bool wordwrap)
java int setConsoleWordWrap( boolean wordwrap)
py def setConsoleWordWrap( wordwrap)
cmd YDisplay target [-layer layerId] 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.

```

js   function setLayerPosition( x, y, scrollTime)
node.js function setLayerPosition( x, y, scrollTime)
php  function setLayerPosition( $x, $y, $scrollTime)
cpp   int setLayerPosition( int x, int y, int scrollTime)
m     -(int) setLayerPosition : (int) x
                  : (int) y
                  : (int) scrollTime
pas   function setLayerPosition( x: LongInt,
                                y: LongInt,
                                scrollTime: LongInt): LongInt
vb    function setLayerPosition( ) As Integer
cs    int setLayerPosition( int x, int y, int scrollTime)
java  int setLayerPosition( int x, int y, int scrollTime)
py    def setLayerPosition( x, y, scrollTime)
cmd   YDisplay target [-layer layerId] setLayerPosition x y scrollTime

```

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.

js	function <b>unhide( )</b>
node.js	function <b>unhide( )</b>
php	function <b>unhide( )</b>
cpp	int <b>unhide( )</b>
m	- <b>(int) unhide</b>
pas	function <b>unhide( ): LongInt</b>
vb	function <b>unhide( ) As Integer</b>
cs	int <b>unhide( )</b>
java	int <b>unhide( )</b>
py	def <b>unhide( )</b>
cmd	<b>YDisplay target [-layer layerId] unhide</b>

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>
nodejs 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()**

Retourne le nom logique du contrôle d'alimentation.

**dualpower→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**dualpower→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**dualpower→get\_powerControl()**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

**dualpower→get\_powerState()**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

**dualpower→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**dualpower→isOnline()**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

**dualpower→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

**dualpower→load(msValidity)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

**dualpower→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

**dualpower→nextDualPower()**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de yFirstDualPower( ).

**dualpower→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**dualpower→set\_logicalName(newval)**

Modifie le nom logique du contrôle d'alimentation.

**dualpower→set\_powerControl(newval)**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

**dualpower→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**dualpower→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YDualPower.FindDualPower()****YDualPower****yFindDualPower()YDualPower.FindDualPower()**

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

js	function <b>yFindDualPower( func)</b>
node.js	function <b>FindDualPower( func)</b>
php	function <b>yFindDualPower( \$func)</b>
cpp	YDualPower* <b>yFindDualPower( const string&amp; func)</b>
m	YDualPower* <b>yFindDualPower( NSString* func)</b>
pas	function <b>yFindDualPower( func: string): TYDualPower</b>
vb	function <b>yFindDualPower( ByVal func As String) As YDualPower</b>
cs	YDualPower <b>FindDualPower( string func)</b>
java	YDualPower <b>FindDualPower( String func)</b>
py	def <b>FindDualPower( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'alimentation soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDualPower.isOnline()` pour tester si le contrôle d'alimentation est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le contrôle d'alimentation sans ambiguïté

**Retourne :**

un objet de classe `YDualPower` qui permet ensuite de contrôler le contrôle d'alimentation.

**YDualPower.FirstDualPower()****YDualPower****yFirstDualPower()YDualPower.FirstDualPower()**

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

js	function <b>yFirstDualPower( )</b>
node.js	function <b>FirstDualPower( )</b>
php	function <b>yFirstDualPower( )</b>
cpp	YDualPower* <b>yFirstDualPower( )</b>
m	YDualPower* <b>yFirstDualPower( )</b>
pas	function <b>yFirstDualPower( ): TYDualPower</b>
vb	function <b>yFirstDualPower( ) As YDualPower</b>
cs	YDualPower <b>FirstDualPower( )</b>
java	YDualPower <b>FirstDualPower( )</b>
py	def <b>FirstDualPower( )</b>

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

**Retourne :**

un pointeur sur un objet `YDualPower`, correspondant à le premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

**dualpower→describe()dualpower.describe()****YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format TYPE (NAME )=SERIAL.FUNCTIONID.

js	function <b>describe()</b>
nodejs	function <b>describe()</b>
php	function <b>describe()</b>
cpp	string <b>describe()</b>
m	-(NSString*) <b>describe</b>
pas	function <b>describe()</b> : string
vb	function <b>describe()</b> As String
cs	string <b>describe()</b>
java	String <b>describe()</b>
py	def <b>describe()</b>

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 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YDualPower <b>target get_advertisedValue</b>

**Retourne :**

une chaîne de caractères représentant la valeur courante du contrôle d'alimentation (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne **Y\_ADVERTISEDVALUE\_INVALID**.

**dualpower→get\_errorMessage()**  
**dualpower→errorMessage()**  
**dualpower.get\_errorMessage()****YDualPower**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
js function get_errorMessage( )
nodejs function get_errorMessage( )
php function get_errorMessage( )
cpp string get_errorMessage( )
m -(NSString*) errorMessage
pas function get_errorMessage( ): string
vb function get_errorMessage( ) As String
cs string get_errorMessage( )
java String get_errorMessage( )
py def get_errorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

```
js function get_extVoltage( )  
node.js function get_extVoltage( )  
php function get_extVoltage( )  
cpp int get_extVoltage( )  
m -(int) extVoltage  
pas function get_extVoltage( ): LongInt  
vb function get_extVoltage( ) As Integer  
cs int get_extVoltage( )  
java int get_extVoltage( )  
py def get_extVoltage( )  
cmd YDualPower target 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.

```
js function get_friendlyName( )
nodejs function get_friendlyName( )
php function get_friendlyName( )
cpp string get_friendlyName( )
m -(NSString*) friendlyName
cs string get_friendlyName( )
java String get_friendlyName( )
py def get_friendlyName( )
```

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) <b>functionDescriptor</b>
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**dualpower→get\_functionId()****YDualPower****dualpower→functionId()dualpower.get\_functionId()**

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

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.

js	function <b>get_logicalName( )</b>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( )</b> : string
vb	function <b>get_logicalName( )</b> As String
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	YDualPower <b>target get_logicalName</b>

**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.

```
js function get_module()
node.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`

**dualpower→get\_module\_async()**  
**dualpower→module\_async()****YDualPower**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

**dualpower→get\_powerControl()**  
**dualpower→powerControl()**  
**dualpower.get\_powerControl()****YDualPower**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
js function get_powerControl( )
nodejs function get_powerControl( )
php function get_powerControl( )
cpp Y_POWERCONTROL_enum get_powerControl( )
m -(Y_POWERCONTROL_enum) powerControl
pas function get_powerControl( ): Integer
vb function get_powerControl( ) As Integer
cs int get_powerControl( )
java int get_powerControl( )
py def get_powerControl( )
cmd YDualPower target 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.

js	function <b>get_powerState( )</b>
nodejs	function <b>get_powerState( )</b>
php	function <b>get_powerState( )</b>
cpp	Y_POWERSTATE_enum <b>get_powerState( )</b>
m	-(Y_POWERSTATE_enum) powerState
pas	function <b>get_powerState( )</b> : Integer
vb	function <b>get_powerState( )</b> As Integer
cs	int <b>get_powerState( )</b>
java	int <b>get_powerState( )</b>
py	def <b>get_powerState( )</b>
cmd	YDualPower <b>target get_powerState</b>

**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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData) {  
m -(void*) userData  
pas function get(userData): Tobject  
vb function get(userData) As Object  
cs object get(userData)  
java Object get(userData)  
py def get(userData)
```

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.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()****YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

<code>js</code>	<code>function nextDualPower( )</code>
<code>nodejs</code>	<code>function nextDualPower( )</code>
<code>php</code>	<code>function nextDualPower( )</code>
<code>cpp</code>	<code>YDualPower * nextDualPower( )</code>
<code>m</code>	<code>-(YDualPower*) nextDualPower</code>
<code>pas</code>	<code>function nextDualPower( ): TYDualPower</code>
<code>vb</code>	<code>function nextDualPower( ) As YDualPower</code>
<code>cs</code>	<code>YDualPower nextDualPower( )</code>
<code>java</code>	<code>YDualPower nextDualPower( )</code>
<code>py</code>	<code>def nextDualPower( )</code>

**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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YDualPowerValueCallback callback)
m -(int) registerValueCallback : (YDualPowerValueCallback) callback
pas function registerValueCallback( callback: TYDualPowerValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YDualPower target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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.

```
js function set_powerControl( newval)
nodejs function set_powerControl( newval)
php function set_powerControl( $newval)
cpp int set_powerControl( Y_POWERCONTROL_enum newval)
m -(int) setPowerControl : (Y_POWERCONTROL_enum) newval
pas function set_powerControl( newval: Integer): integer
vb function set_powerControl( ByVal newval As Integer) As Integer
cs int set_powerControl( int newval)
java int set_powerControl( int newval)
py def set_powerControl( newval)
cmd YDualPower target 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).

js	function set(userData( data)
node.js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

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()****YDualPower**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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>
node.js 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()

Rétabli le système de fichier dans on état original, défragmenté.

#### files→get\_advertisedValue()

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

#### files→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### files→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### files→get\_filesCount()

Retourne le nombre de fichiers présents dans le système de fichier.

#### files→get\_freeSpace()

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

#### files→get\_friendlyName()

Retourne un identifiant global du système de fichier au format NOM\_MODULE.NOM\_FONCTION.

#### files→get\_functionDescriptor()

### 3. Reference

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>files→get_functionId()</b> Retourne l'identifiant matériel du système de fichier, sans référence au module.
<b>files→get_hardwareId()</b> Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.
<b>files→get_list(pattern)</b> Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.
<b>files→get_logicalName()</b> Retourne le nom logique du système de fichier.
<b>files→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>files→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>files→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>files→isOnline()</b> Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.
<b>files→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.
<b>files→load(msValidity)</b> Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.
<b>files→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.
<b>files→nextFiles()</b> Continue l'énumération des système de fichier commencée à l'aide de yFirstFiles( ).
<b>files→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>files→remove(pathname)</b> Efface un fichier, spécifié par son path complet, du système de fichier.
<b>files→set_logicalName(newval)</b> Modifie le nom logique du système de fichier.
<b>files→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>files→upload(pathname, content)</b> Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.
<b>files→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.

**YFiles.FindFiles()****YFiles****yFindFiles()YFiles.FindFiles()**

Permet de retrouver un système de fichier d'après un identifiant donné.

js	function <b>yFindFiles( func)</b>
node.js	function <b>FindFiles( func)</b>
php	function <b>yFindFiles( \$func)</b>
cpp	YFiles* <b>yFindFiles( string func)</b>
m	+( <b>YFiles*</b> ) <b>yFindFiles : (NSString*) func</b>
pas	function <b>yFindFiles( func: string): TYFiles</b>
vb	function <b>yFindFiles( ByVal func As String) As YFiles</b>
cs	<b>YFiles FindFiles( string func)</b>
java	<b>YFiles FindFiles( String func)</b>
py	def <b>FindFiles( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnLine()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le système de fichier sans ambiguïté

**Retourne :**

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

**YFiles.FirstFiles()****YFiles****yFirstFiles()YFiles.FirstFiles()**

Commence l'énumération des système de fichier accessibles par la librairie.

js	function <b>yFirstFiles()</b>
node.js	function <b>FirstFiles()</b>
php	function <b>yFirstFiles()</b>
cpp	YFiles* <b>yFirstFiles()</b>
m	YFiles* <b>yFirstFiles()</b>
pas	function <b>yFirstFiles()</b> : TYFiles
vb	function <b>yFirstFiles()</b> As YFiles
cs	YFiles <b>FirstFiles()</b>
java	YFiles <b>FirstFiles()</b>
py	def <b>FirstFiles()</b>

Utiliser la fonction `YFiles.nextFiles()` pour itérer sur les autres système de fichier.

**Retourne :**

un pointeur sur un objet `YFiles`, correspondant à le 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.

js	function <b>describe</b> ( )
node.js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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 système de fichier (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**files→download()**

YFiles

Télécharge le fichier choisi du filesystème et retourne son contenu.

```
js function download( pathname)
nodejs function download( pathname)
php function download( $pathname)
cpp string download( string pathname)
m -(NSData*) download : (NSString*) pathname
pas function download( pathname: string): TByteArray
vb function download( ) As Byte
py def download( pathname)
cmd YFiles target 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()****YFiles**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
js function download_async( pathname, callback, context)
nodejs 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**

Rétablissement le système de fichier dans un état original, défragmenté.

js	function <b>format_fs( )</b>
nodejs	function <b>format_fs( )</b>
php	function <b>format_fs( )</b>
cpp	int <b>format_fs( )</b>
m	- <b>(int) format_fs</b>
pas	function <b>format_fs( ): LongInt</b>
vb	function <b>format_fs( ) As Integer</b>
cs	int <b>format_fs( )</b>
java	int <b>format_fs( )</b>
py	def <b>format_fs( )</b>
cmd	YFiles <b>target format_fs</b>

entièrement vide. Tous les fichiers précédemment chargés sont irrémédiablement effacés.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→get\_advertisedValue()****YFiles****files→advertisedValue()files.get\_advertisedValue()**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

js	function <b>get_advertisedValue( )</b>
nodejs	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	<b>YFiles target get_advertisedValue</b>

**Retourne :**

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne **Y\_ADVERTISEDVALUE\_INVALID**.

**YFiles**  
**files→getErrorMessage()**  
**files→errorMessage(files.getErrorMessage())**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
js function getErrorMessage( )
nodejs function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ):string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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()**  
**files→filesCount()files.get\_filesCount()****YFiles**

Retourne le nombre de fichiers présents dans le système de fichier.

```
js function get_filesCount( )
node.js function get_filesCount( )
php function get_filesCount( )
cpp int get_filesCount( )
m -(int) filesCount
pas function get_filesCount( ): LongInt
vb function get_filesCount( ) As Integer
cs int get_filesCount( )
java int get_filesCount( )
py def get_filesCount( )
cmd YFiles target 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.

js	function <b>get_freeSpace()</b>
nodejs	function <b>get_freeSpace()</b>
php	function <b>get_freeSpace()</b>
cpp	int <b>get_freeSpace()</b>
m	-(int) freeSpace
pas	function <b>get_freeSpace()</b> : LongInt
vb	function <b>get_freeSpace()</b> As Integer
cs	int <b>get_freeSpace()</b>
java	int <b>get_freeSpace()</b>
py	def <b>get_freeSpace()</b>
cmd	YFiles <b>target get_freeSpace</b>

**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()**  
**files→friendlyName()files.get\_friendlyName()****YFiles**

Retourne un identifiant global du système de fichier au format NOM\_MODULE.NOM\_FONCTION.

js	function get_friendlyName( )
node.js	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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

**files→get\_functionId()  
files→functionId()files.get\_functionId()****YFiles**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**files→get\_hardwareId()****YFiles****files→hardwareId()files.get\_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId()</b>
nodejs	function <b>get_hardwareId()</b>
php	function <b>get_hardwareId()</b>
cpp	string <b>get_hardwareId()</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId()</b> As String
cs	string <b>get_hardwareId()</b>
java	String <b>get_hardwareId()</b>
py	def <b>get_hardwareId()</b>

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() files→list()files.get\_list()

YFiles

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
js function get_list( pattern)
nodejs function get_list( pattern)
php function get_list( $pattern)
cpp vector<YFileRecord> get_list( string pattern)
m -(NSMutableArray*) list : (NSString*) pattern
pas function get_list( pattern: string): TYFileRecordArray
vb function get_list( ) As List
cs List<YFileRecord> get_list( string pattern)
java ArrayList<YFileRecord> get_list( String pattern)
py def get_list( pattern)
cmd YFiles target 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.

**files→get\_logicalName()****YFiles****files→logicalName()files.get\_logicalName()**

Retourne le nom logique du système de fichier.

```
js function get_logicalName( )
nodejs function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YFiles target get_logicalName
```

**Retourne :**

une chaîne de caractères représentant le nom logique du système de fichier. En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

---

<b>files→get_module()</b>	<b>YFiles</b>
<b>files→module()files.get_module()</b>	

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module( )
node.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`

## files→get\_module\_async() files→module\_async()

YFiles

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context )
nodejs function get_module_async( callback, context )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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)**  
**files→userData(files.get(userData))****YFiles**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData)  
m -(void*) userData  
pas function get(userData): Tobject  
vb function get(userData) As Object  
cs object get(userData)  
java Object get(userData)  
py def get(userData)
```

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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	<b>def isOnline( )</b>

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()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()****YFiles**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextFiles()</b>
nodejs	function <b>nextFiles()</b>
php	function <b>nextFiles()</b>
cpp	YFiles * <b>nextFiles()</b>
m	-(YFiles*) <b>nextFiles</b>
pas	function <b>nextFiles()</b> : TYFiles
vb	function <b>nextFiles()</b> As YFiles
cs	YFiles <b>nextFiles()</b>
java	YFiles <b>nextFiles()</b>
py	def <b>nextFiles()</b>

**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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YFilesValueCallback callback)
m -(int) registerValueCallback : (YFilesValueCallback) callback
pas function registerValueCallback( callback: TYFilesValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

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.

js	function <b>remove( pathname)</b>
node.js	function <b>remove( pathname)</b>
php	function <b>remove( \$pathname)</b>
cpp	int <b>remove( string pathname)</b>
m	- <b>(int) remove : (NSString*) pathname</b>
pas	function <b>remove( pathname: string): LongInt</b>
vb	function <b>remove( ) As Integer</b>
cs	int <b>remove( string pathname)</b>
java	int <b>remove( String pathname)</b>
py	def <b>remove( pathname)</b>
cmd	YFiles <b>target remove pathname</b>

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.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YFiles 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 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).

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	- <b>(void) set(userData : (void*) data)</b>
pas	procedure <b>set(userData Tobject data)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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.

```
js function upload( pathname, content)
nodejs function upload( pathname, content)
php function upload( $pathname, $content)
cpp int upload( string pathname, string content)
m -(int) upload : (NSString*) pathname
               : (NSData*) content
pas function upload( pathname: string, content: TByteArray): LongInt
vb procedure upload( )
cs int upload( string pathname)
java int upload( String pathname)
py def upload( pathname, content)
cmd YFiles target 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()****YFiles**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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 extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_genericsensor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGenericSensor = yoctolib.YGenericSensor;
require_once('yocto_genericsensor.php');
#include "yocto_genericsensor.h"
m #import "yocto_genericsensor.h"
pas uses yocto_genericsensor;
vb yocto_genericsensor.vb
cs yocto_genericsensor.cs
java import com.yoctopuce.YoctoAPI.YGenericSensor;
py from yocto_genericsensor import *

```

### Fonction globales

#### yFindGenericSensor(func)

Permet de retrouver un capteur générique d'après un identifiant donné.

#### yFirstGenericSensor()

Commence l'énumération des capteurs génériques accessibles par la librairie.

### Méthodes des objets YGenericSensor

#### genericsensor→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### genericsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### genericsensor→get\_advertisedValue()

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

#### genericsensor→get\_currentRawValue()

Retourne la valeur brute rentrée par le capteur (sans arrondi ni calibration).

#### genericsensor→get\_currentValue()

Retourne la valeur mesurée actuelle.

#### genericsensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### genericsensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### genericsensor→get\_friendlyName()

Retourne un identifiant global du capteur générique au format NOM\_MODULE . NOM\_FONCTION.

#### genericsensor→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### genericsensor→get\_functionId()

Retourne l'identifiant matériel du capteur générique, sans référence au module.

#### genericsensor→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur générique au format SERIAL . FUNCTIONID.

#### **genericsensor→get\_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

#### **genericsensor→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

#### **genericsensor→get\_logicalName()**

Retourne le nom logique du capteur générique.

#### **genericsensor→get\_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

#### **genericsensor→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **genericsensor→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **genericsensor→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

#### **genericsensor→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

#### **genericsensor→get\_resolution()**

Retourne la résolution des valeurs mesurées.

#### **genericsensor→get\_signalRange()**

Retourne la plage de signal électrique utilisée par le capteur.

#### **genericsensor→get\_signalUnit()**

Retourne l'unité du signal électrique utilisée par le capteur.

#### **genericsensor→get\_signalValue()**

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

#### **genericsensor→get\_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

#### **genericsensor→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **genericsensor→get\_valueRange()**

Retourne la plage de valeurs physiques mesurés par le capteur.

#### **genericsensor→isOnline()**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

#### **genericsensor→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

#### **genericsensor→load(msValidity)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

#### **genericsensor→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **genericsensor→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

#### **genericsensor→nextGenericSensor()**

### 3. Reference

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

#### `genericSensor->registerTimedReportCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### `genericSensor->registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### `genericSensor->set_highestValue(newval)`

Modifie la mémoire de valeur maximale observée.

#### `genericSensor->set_logFrequency(newval)`

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### `genericSensor->set_logicalName(newval)`

Modifie le nom logique du capteur générique.

#### `genericSensor->set_lowestValue(newval)`

Modifie la mémoire de valeur minimale observée.

#### `genericSensor->set_reportFrequency(newval)`

Modifie la fréquence de notification périodique des valeurs mesurées.

#### `genericSensor->set_resolution(newval)`

Modifie la résolution des valeurs physique mesurées.

#### `genericSensor->set_signalRange(newval)`

Modifie la plage de signal électrique utilisée par le capteur.

#### `genericSensor->set_unit(newval)`

Change l'unité dans laquelle la valeur mesurée est exprimée.

#### `genericSensor->set_userData(data)`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

#### `genericSensor->set_valueRange(newval)`

Modifie la plage de valeurs physiques mesurés par le capteur.

#### `genericSensor->wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YGenericSensor.FindGenericSensor() yFindGenericSensor() YGenericSensor.FindGenericSensor()

## YGenericSensor

Permet de retrouver un capteur générique d'après un identifiant donné.

<code>js</code>	<code>function yFindGenericSensor( func)</code>
<code>node.js</code>	<code>function FindGenericSensor( func)</code>
<code>php</code>	<code>function yFindGenericSensor( \$func)</code>
<code>cpp</code>	<code>YGenericSensor* yFindGenericSensor( const string&amp; func)</code>
<code>m</code>	<code>YGenericSensor* yFindGenericSensor( NSString* func)</code>
<code>pas</code>	<code>function yFindGenericSensor( func: string): TYGenericSensor</code>
<code>vb</code>	<code>function yFindGenericSensor( ByVal func As String) As YGenericSensor</code>
<code>cs</code>	<code>YGenericSensor FindGenericSensor( string func)</code>
<code>java</code>	<code>YGenericSensor FindGenericSensor( String func)</code>
<code>py</code>	<code>def FindGenericSensor( 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 le capteur générique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGenericSensor.isOnline()` pour tester si le capteur générique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le capteur générique sans ambiguïté

### Retourne :

un objet de classe `YGenericSensor` qui permet ensuite de contrôler le capteur générique.

## YGenericSensor.FirstGenericSensor() yFirstGenericSensor() YGenericSensor.FirstGenericSensor()

**YGenericSensor**

Commence l'énumération des capteurs génériques accessibles par la librairie.

```
js function yFirstGenericSensor( )  
nodejs function FirstGenericSensor( )  
php function yFirstGenericSensor( )  
cpp YGenericSensor* yFirstGenericSensor( )  
m YGenericSensor* yFirstGenericSensor( )  
pas function yFirstGenericSensor( ): TYGenericSensor  
vb function yFirstGenericSensor( ) As YGenericSensor  
cs YGenericSensor FirstGenericSensor( )  
java YGenericSensor FirstGenericSensor( )  
py def FirstGenericSensor( )
```

Utiliser la fonction `YGenericSensor.nextGenericSensor()` pour itérer sur les autres capteurs génériques.

**Retourne :**

un pointeur sur un objet `YGenericSensor`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)

m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YGenericSensor target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

<b>js</b>	function <b>get_advertisedValue( )</b>
<b>nodejs</b>	function <b>get_advertisedValue( )</b>
<b>php</b>	function <b>get_advertisedValue( )</b>
<b>cpp</b>	string <b>get_advertisedValue( )</b>
<b>m</b>	-(NSString*) <b>advertisedValue</b>
<b>pas</b>	function <b>get_advertisedValue( )</b> : string
<b>vb</b>	function <b>get_advertisedValue( )</b> As String
<b>cs</b>	string <b>get_advertisedValue( )</b>
<b>java</b>	String <b>get_advertisedValue( )</b>
<b>py</b>	def <b>get_advertisedValue( )</b>
<b>cmd</b>	YGenericSensor <b>target get_advertisedValue</b>

**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).

```
js function get_currentRawValue( )  
nodejs function get_currentRawValue( )  
php function get_currentRawValue( )  
cpp double get_currentRawValue( )  
m -(double) currentRawValue  
pas function get_currentRawValue( ): double  
vb function get_currentRawValue( ) As Double  
cs double get_currentRawValue( )  
java double get_currentRawValue( )  
py def get_currentRawValue( )  
cmd YGenericSensor target 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.

js	function <b>get_currentValue( )</b>
node.js	function <b>get_currentValue( )</b>
php	function <b>get_currentValue( )</b>
cpp	double <b>get_currentValue( )</b>
m	-(double) <b>currentValue</b>
pas	function <b>get_currentValue( )</b> : double
vb	function <b>get_currentValue( )</b> As Double
cs	double <b>get_currentValue( )</b>
java	double <b>get_currentValue( )</b>
py	def <b>get_currentValue( )</b>
cmd	YGenericSensor <b>target get_currentValue</b>

**Retourne :**

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**genericsensor→get\_errorMessage()**  
**genericsensor→errorMessage()**  
**genericsensor.get\_errorMessage()**

**YGenericSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

```
js function get_errorMessage( )  
nodejs function get_errorMessage( )  
php function get_errorMessage( )  
cpp string get_errorMessage( )  
m -(NSString*) errorMessage  
pas function get_errorMessage( ): string  
vb function get_errorMessage( ) As String  
cs string get_errorMessage( )  
java String get_errorMessage( )  
py def get_errorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) <b>functionDescriptor</b>
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	def <b>get_functionDescriptor()</b>

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**genericsensor→get\_functionId()**  
**genericsensor→functionId()**  
**genericsensor.get\_functionId()**

**YGenericSensor**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

js	function <b>get_functionId( )</b>
nodejs	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( )</b> As String
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

js	function <b>get_hardwareId()</b>
node.js	function <b>get_hardwareId()</b>
php	function <b>get_hardwareId()</b>
cpp	string <b>get_hardwareId()</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId()</b> As String
cs	string <b>get_hardwareId()</b>
java	String <b>get_hardwareId()</b>
py	def <b>get_hardwareId()</b>

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.

js	function <b>get_highestValue( )</b>
nodejs	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YGenericSensor <b>target get_highestValue</b>

**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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YGenericSensor target 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.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YGenericSensor target 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.

js	function <b>get_lowestValue( )</b>
node.js	function <b>get_lowestValue( )</b>
php	function <b>get_lowestValue( )</b>
cpp	double <b>get_lowestValue( )</b>
m	-(double) lowestValue
pas	function <b>get_lowestValue( ): double</b>
vb	function <b>get_lowestValue( ) As Double</b>
cs	double <b>get_lowestValue( )</b>
java	double <b>get_lowestValue( )</b>
py	def <b>get_lowestValue( )</b>
cmd	YGenericSensor <b>target get_lowestValue</b>

**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.

js	function <b>get_module()</b>
nodejs	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As <code>YModule</code>
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YGenericSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

**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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m     -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime
pas   function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb    function get_recordedData( ) As YDataSet
cs    YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YGenericSensor target 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.

**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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YGenericSensor target 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.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YGenericSensor target 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.

js	function <b>get_signalRange( )</b>
node.js	function <b>get_signalRange( )</b>
php	function <b>get_signalRange( )</b>
cpp	string <b>get_signalRange( )</b>
m	-(NSString*) <b>signalRange</b>
pas	function <b>get_signalRange( )</b> : string
vb	function <b>get_signalRange( )</b> As String
cs	string <b>get_signalRange( )</b>
java	String <b>get_signalRange( )</b>
py	def <b>get_signalRange( )</b>
cmd	YGenericSensor <b>target get_signalRange</b>

**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.

js	function <b>get_signalUnit( )</b>
nodejs	function <b>get_signalUnit( )</b>
php	function <b>get_signalUnit( )</b>
cpp	string <b>get_signalUnit( )</b>
m	-(NSString*) signalUnit
pas	function <b>get_signalUnit( )</b> : string
vb	function <b>get_signalUnit( )</b> As String
cs	string <b>get_signalUnit( )</b>
java	String <b>get_signalUnit( )</b>
py	def <b>get_signalUnit( )</b>
cmd	YGenericSensor <b>target get_signalUnit</b>

**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.

js	function <b>get_signalValue( )</b>
node.js	function <b>get_signalValue( )</b>
php	function <b>get_signalValue( )</b>
cpp	double <b>get_signalValue( )</b>
m	-(double) <b>signalValue</b>
pas	function <b>get_signalValue( )</b> : double
vb	function <b>get_signalValue( )</b> As Double
cs	double <b>get_signalValue( )</b>
java	double <b>get_signalValue( )</b>
py	<b>def get_signalValue( )</b>
cmd	YGenericSensor <b>target get_signalValue</b>

**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.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	YGenericSensor <b>target get_unit</b>

**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).

js	function <b>get(userData)</b>
node.js	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	def <b>get(userData)</b>

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.

```
js function get_valueRange( )  
nodejs function get_valueRange( )  
php function get_valueRange( )  
cpp string get_valueRange( )  
m -(NSString*) valueRange  
pas function get_valueRange( ): string  
vb function get_valueRange( ) As String  
cs string get_valueRange( )  
java String get_valueRange( )  
py def get_valueRange( )  
cmd YGenericSensor target 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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	<b>def isOnline( )</b>

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()****YGenericSensor**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function loadCalibrationPoints( var rawValues: TDoubleArray,
                                      var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( )
cs    int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
py    def loadCalibrationPoints( rawValues, refValues)
cmd   YGenericSensor target loadCalibrationPoints rawValues refValues

```

### 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()

## YGenericSensor

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextGenericSensor()</b>
nodejs	function <b>nextGenericSensor()</b>
php	function <b>nextGenericSensor()</b>
cpp	YGenericSensor * <b>nextGenericSensor()</b>
m	- <b>(YGenericSensor*) nextGenericSensor</b>
pas	function <b>nextGenericSensor()</b> : TYGenericSensor
vb	function <b>nextGenericSensor()</b> As YGenericSensor
cs	YGenericSensor <b>nextGenericSensor()</b>
java	YGenericSensor <b>nextGenericSensor()</b>
py	def <b>nextGenericSensor()</b>

**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.

js	function registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YGenericSensorTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YGenericSensorTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYGenericSensorTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**genericsensor→registerValueCallback()  
genericsensor.registerValueCallback()****YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YGenericSensorValueCallback callback)
m -(int) registerValueCallback : (YGenericSensorValueCallback) callback
pas function registerValueCallback( callback: TYGenericSensorValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YGenericSensor target set_highestValue newval
```

**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.

```
js function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YGenericSensor target 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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YGenericSensor target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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.

```
js function set_lowestValue( newval)
nodejs function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YGenericSensor target set_lowestValue newval
```

**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.

js	function <b>set_reportFrequency( newval)</b>
node.js	function <b>set_reportFrequency( newval)</b>
php	function <b>set_reportFrequency( \$newval)</b>
cpp	int <b>set_reportFrequency( const string&amp; newval)</b>
m	-(int) <b>setReportFrequency : (NSString*) newval</b>
pas	function <b>set_reportFrequency( newval: string): integer</b>
vb	function <b>set_reportFrequency( ByVal newval As String) As Integer</b>
cs	int <b>set_reportFrequency( string newval)</b>
java	int <b>set_reportFrequency( String newval)</b>
py	def <b>set_reportFrequency( newval)</b>
cmd	YGenericSensor <b>target set_reportFrequency newval</b>

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.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YGenericSensor target 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.

<b>js</b>	function <b>set_signalRange( newval)</b>
<b>node.js</b>	function <b>set_signalRange( newval)</b>
<b>php</b>	function <b>set_signalRange( \$newval)</b>
<b>cpp</b>	int <b>set_signalRange( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setSignalRange : (NSString*) newval</b>
<b>pas</b>	function <b>set_signalRange( newval: string): integer</b>
<b>vb</b>	function <b>set_signalRange( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_signalRange( string newval)</b>
<b>java</b>	int <b>set_signalRange( String newval)</b>
<b>py</b>	def <b>set_signalRange( newval)</b>
<b>cmd</b>	<b>YGenericSensor target set_signalRange newval</b>

**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.

js	function <b>set_unit( newval)</b>
node.js	function <b>set_unit( newval)</b>
php	function <b>set_unit( \$newval)</b>
cpp	int <b>set_unit( const string&amp; newval)</b>
m	-(int) setUnit : (NSString*) <b>newval</b>
pas	function <b>set_unit( newval: string): integer</b>
vb	function <b>set_unit( ByVal newval As String) As Integer</b>
cs	int <b>set_unit( string newval)</b>
java	int <b>set_unit( String newval)</b>
py	def <b>set_unit( newval)</b>
cmd	<b>YGenericSensor target set_unit newval</b>

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).

```
js function setUserData( data)
node.js function setUserData( data)
php function setUserData( $data)
cpp void setUserData( void* data)
m -(void) setUserData : (void*) data
pas procedure setUserData( data: Tobject)
vb procedure setUserData( ByVal data As Object)
cs void setUserData( object data)
java void setUserData( Object data)
py def setUserData( data)
```

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.

```
js function set_valueRange( newval)
nodejs function set_valueRange( newval)
php function set_valueRange( $newval)
cpp int set_valueRange( const string& newval)
m -(int) setValueRange : (NSString*) newval
pas function set_valueRange( newval: string): integer
vb function set_valueRange( ByVal newval As String) As Integer
cs int set_valueRange( string newval)
java int set_valueRange( String newval)
py def set_valueRange( newval)
cmd YGenericSensor target set_valueRange newval
```

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()

## YGenericSensor

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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 extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_gyro.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGyro = yoctolib.YGyro;
php require_once('yocto_gyro.php');
cpp #include "yocto_gyro.h"
m #import "yocto_gyro.h"
pas uses yocto_gyro;
vb yocto_gyro.vb
cs yocto_gyro.cs
java import com.yoctopuce.YoctoAPI.YGyro;
py from yocto_gyro import *

```

### Fonction globales

#### **yFindGyro(func)**

Permet de retrouver un gyroscope d'après un identifiant donné.

#### **yFirstGyro()**

Commence l'énumération des gyroscopes accessibles par la librairie.

### Méthodes des objets YGyro

#### **gyro→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **gyro→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format TYPE (NAME) = SERIAL . FUNCTIONID.

#### **gyro→get\_advertisedValue()**

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

#### **gyro→get\_currentRawValue()**

Retourne la valeur brute 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.
<b>gyro→get_yValue()</b>	Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.
<b>gyro→get_zValue()</b>	Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.
<b>gyro→isOnline()</b>	Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.
<b>gyro→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.
<b>gyro→load(msValidity)</b>	Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.
<b>gyro→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>gyro→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.
<b>gyro→nextGyro()</b>	Continue l'énumération des gyroscopes commencée à l'aide de yFirstGyro( ).
<b>gyro→registerAnglesCallback(callback)</b>	Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.
<b>gyro→registerQuaternionCallback(callback)</b>	Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.
<b>gyro→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>gyro→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>gyro→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.
<b>gyro→set_logFrequency(newval)</b>	Modifie la fréquence d'enregistrement des mesures dans le datalogger.
<b>gyro→set_logicalName(newval)</b>	Modifie le nom logique du gyroscope.
<b>gyro→set_lowestValue(newval)</b>	Modifie la mémoire de valeur minimale observée.
<b>gyro→set_reportFrequency(newval)</b>	Modifie la fréquence de notification périodique des valeurs mesurées.
<b>gyro→set_resolution(newval)</b>	Modifie la résolution des valeurs physique mesurées.
<b>gyro→set_userData(data)</b>	Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).
<b>gyro→wait_async(callback, context)</b>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YGyro.FindGyro()****YGyro****yFindGyro()YGyro.FindGyro()**

Permet de retrouver un gyroscope d'après un identifiant donné.

js	function <b>yFindGyro( func)</b>
node.js	function <b>FindGyro( func)</b>
php	function <b>yFindGyro( \$func)</b>
cpp	YGyro* <b>yFindGyro( string func)</b>
m	+( <b>YGyro*</b> ) <b>yFindGyro</b> : ( <b>NSString*</b> ) <b>func</b>
pas	function <b>yFindGyro( func: string): TYGyro</b>
vb	function <b>yFindGyro( ByVal func As String) As YGyro</b>
cs	<b>YGyro FindGyro( string func)</b>
java	<b>YGyro FindGyro( String func)</b>
py	def <b>FindGyro( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le gyroscope sans ambiguïté

**Retourne :**

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

## YGyro.FirstGyro() yFirstGyro()YGyro.FirstGyro()

YGyro

Commence l'énumération des gyroscopes accessibles par la librairie.

```
js function yFirstGyro( )
node.js function FirstGyro( )
php function yFirstGyro( )
cpp YGyro* yFirstGyro( )
m YGyro* yFirstGyro( )
pas function yFirstGyro( ): TYGyro
vb function yFirstGyro( ) As YGyro
cs YGyro FirstGyro( )
java YGyro FirstGyro( )
def FirstGyro( )
py
```

Utiliser la fonction `YGyro.nextGyro()` pour itérer sur les autres gyroscopes.

### Retourne :

un pointeur sur un objet `YGyro`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)

m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YGyro target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe()</b>
nodejs	function <b>describe()</b>
php	function <b>describe()</b>
cpp	string <b>describe()</b>
m	-(NSString*) <b>describe</b>
pas	function <b>describe()</b> : string
vb	function <b>describe()</b> As String
cs	string <b>describe()</b>
java	String <b>describe()</b>
py	def <b>describe()</b>

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 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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YGyro target get_advertisedValue
```

**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.

---

<b>gyro→get_currentRawValue()</b>	<b>YGyro</b>
<b>gyro→currentRawValue()gyro.get_currentRawValue()</b>	

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
js function get_currentRawValue( )
node.js function get_currentRawValue( )
php function get_currentRawValue( )
cpp double get_currentRawValue( )
m -(double) currentRawValue
pas function get_currentRawValue( ): double
vb function get_currentRawValue( ) As Double
cs double get_currentRawValue( )
java double get_currentRawValue( )
py def get_currentRawValue( )
cmd YGyro target get_currentRawValue
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

## gyro→get\_currentValue() gyro→currentValue()gyro.get\_currentValue()

YGyro

Retourne la valeur actuelle de la vitesse angulaire.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YGyro target get_currentValue
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la vitesse angulaire

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**gyro→getErrorMessage()****YGyro****gyro→errorMessage()gyro.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
js function getErrorMessage( )  
node.js function getErrorMessage( )  
php function getErrorMessage( )  
cpp string getErrorMessage( )  
m -(NSString*) errorMessage  
pas function getErrorMessage( ): string  
vb function getErrorMessage( ) As String  
cs string getErrorMessage( )  
java String getErrorMessage( )  
py def getErrorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

**gyro→get\_errorType()****YGyro****gyro→errorType()gyro.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

---

<b>gyro→get_friendlyName()</b>	<b>YGyro</b>
<b>gyro→friendlyName()gyro.get_friendlyName()</b>	

---

Retourne un identifiant global du gyroscope au format NOM\_MODULE . NOM\_FONCTION.

```
js   function get_friendlyName( )  
node.js function get_friendlyName( )  
php  function get_friendlyName( )  
cpp   string get_friendlyName( )  
m    -(NSString*) friendlyName  
cs   string get_friendlyName( )  
java  String get_friendlyName( )  
py   def get_friendlyName( )
```

Le chaîne renvoyée utilise soit les noms logiques du module et du gyroscope si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du gyroscope (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant le gyroscope en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**gyro→get\_functionDescriptor()**  
**gyro→functionDescriptor()**  
**gyro.get\_functionDescriptor()****YGyro**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) <b>functionDescriptor</b>
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	def <b>get_functionDescriptor()</b>

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

---

<b>gyro→get_functionId()</b>	<b>YGyro</b>
<b>gyro→functionId()gyro.get_functionId()</b>	

---

Retourne l'identifiant matériel du gyroscope, sans référence au module.

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**gyro→get.hardwareId()****YGyro****gyro→hardwareId()gyro.get.hardwareId()**

Retourne l'identifiant matériel unique du gyroscope au format SERIAL.FUNCTIONID.

js	function <b>get.hardwareId( )</b>
nodejs	function <b>get.hardwareId( )</b>
php	function <b>get.hardwareId( )</b>
cpp	string <b>get.hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get.hardwareId( ) As String</b>
cs	string <b>get.hardwareId( )</b>
java	String <b>get.hardwareId( )</b>
py	def <b>get.hardwareId( )</b>

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**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.

```
js function get_heading( )
nodejs function get_heading( )
php function get_heading( )
cpp double get_heading( )
m -(double) heading
pas function get_heading( ): double
vb function get_heading( ) As Double
cs double get_heading( )
java double get_heading( )
py def get_heading( )
```

L'axe de lacet peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

**Retourne :**

un nombre à virgule correspondant au cap, exprimé en degrés (entre 0 et 360).

**gyro→get\_highestValue()****YGyro****gyro→highestValue()gyro.get\_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	<b>YGyro target get_highestValue</b>

**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.

```
js function get_logFrequency( )  
nodejs function get_logFrequency( )  
php function get_logFrequency( )  
cpp string get_logFrequency( )  
m -(NSString*) logFrequency  
pas function get_logFrequency( ):string  
vb function get_logFrequency( ) As String  
cs string get_logFrequency( )  
java String get_logFrequency( )  
py def get_logFrequency( )  
cmd YGyro target 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

---

Retourne le nom logique du gyroscope.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YGyro target 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()**  
**gyro→lowestValue()gyro.get\_lowestValue()****YGyro**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

```
js function get_lowestValue( )  
node.js function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YGyro target 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() gyro→module()gyro.get\_module()

YGyro

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module( )</b>
nodejs	function <b>get_module( )</b>
php	function <b>get_module( )</b>
cpp	YModule * <b>get_module( )</b>
m	-(YModule*) module
pas	function <b>get_module( ): TYModule</b>
vb	function <b>get_module( ) As YModule</b>
cs	YModule <b>get_module( )</b>
java	YModule <b>get_module( )</b>
py	def <b>get_module( )</b>

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule rentrée ne sera pas joignable.

**Retourne :**

une instance de YModule

**gyro→get\_module\_async()**  
**gyro→module\_async()****YGyro**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context )
node.js 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.

**gyro→get\_pitch()****YGyro****gyro→pitch()gyro.get\_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
js function get_pitch( )
nodejs function get_pitch( )
php function get_pitch( )
cpp double get_pitch( )
m -(double) pitch
pas function get_pitch( ): double
vb function get_pitch( ) As Double
cs double get_pitch( )
java double get_pitch( )
py def get_pitch( )
```

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.

```
js function get_quaternionW( )
nodejs function get_quaternionW( )
php function get_quaternionW( )
cpp double get_quaternionW( )
m -(double) quaternionW
pas function get_quaternionW(): double
vb function get_quaternionW( ) As Double
cs double get_quaternionW( )
java double get_quaternionW( )
py def get_quaternionW( )
```

**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.

```
js function get_quaternionX( )
nodejs function get_quaternionX( )
php function get_quaternionX( )
cpp double get_quaternionX( )
m -(double) quaternionX
pas function get_quaternionX( ): double
vb function get_quaternionX( ) As Double
cs double get_quaternionX( )
java double get_quaternionX( )
py def get_quaternionX( )
```

La composante x est essentiellement corrélée aux rotations sur l'axe de roulis.

**Retourne :**

un nombre à virgule correspondant à la composante x du quaternion.

**gyro→get\_quaternionY()**  
**gyro→quaternionY()gyro.get\_quaternionY()****YGyro**

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.

```
js function get_quaternionY( )
nodejs function get_quaternionY( )
php function get_quaternionY( )
cpp double get_quaternionY( )
m -(double) quaternionY
pas function get_quaternionY( ): double
vb function get_quaternionY( ) As Double
cs double get_quaternionY( )
java double get_quaternionY( )
py def get_quaternionY( )
```

La composante  $y$  est essentiellement corrélée aux rotations sur l'axe de tangage.

**Retourne :**

un nombre à virgule correspondant à la composante  $y$  du quaternion.

**gyro→get\_quaternionZ()****YGyro****gyro→quaternionZ()gyro.get\_quaternionZ()**

Retourne la composante  $z$  du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
js function get_quaternionZ( )  
nodejs function get_quaternionZ( )  
php function get_quaternionZ( )  
cpp double get_quaternionZ( )  
m -(double) quaternionZ  
pas function get_quaternionZ( ): double  
vb function get_quaternionZ( ) As Double  
cs double get_quaternionZ( )  
java double get_quaternionZ( )  
py def get_quaternionZ( )
```

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() gyro→recordedData()gyro.get\_recordedData()

YGyro

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php   function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m     -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime
pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java YDataSet get_recordedData( long startTime, long endTime)
py   def get_recordedData( startTime, endTime)
cmd  YGyro target 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.

**gyro→get\_reportFrequency()****YGyro****gyro→reportFrequency()gyro.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YGyro target get_reportFrequency
```

**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()**  
**gyro→resolution()gyro.get\_resolution()****YGyro**

Retourne la résolution des valeurs mesurées.

```
js function get_resolution( )
node.js function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YGyro target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**gyro→get\_roll()****YGyro****gyro→roll()gyro.get\_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
js function get_roll( )
nodejs function get_roll( )
php function get_roll( )
cpp double get_roll( )
m -(double) roll
pas function get_roll( ): double
vb function get_roll( ) As Double
cs double get_roll( )
java double get_roll( )
py def get_roll( )
```

L'axe de roulis peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

**Retourne :**

un nombre à virgule correspondant à l'inclinaison, exprimée en degrés (entre -180 et +180).

**gyro→get\_unit()**  
**gyro→unit()gyro.get\_unit()****YGyro**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YGyro target get_unit</b>

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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()**  
**gyro→xValue()gyro.get\_xValue()****YGyro**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

js	function <b>get_xValue( )</b>
node.js	function <b>get_xValue( )</b>
php	function <b>get_xValue( )</b>
cpp	double <b>get_xValue( )</b>
m	-(double) xValue
pas	function <b>get_xValue( )</b> : double
vb	function <b>get_xValue( )</b> As Double
cs	double <b>get_xValue( )</b>
java	double <b>get_xValue( )</b>
py	def <b>get_xValue( )</b>

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_XVALUE\_INVALID**.

## gyro→get\_yValue() gyro→yValue()gyro.get\_yValue()

YGyro

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

```
js function get_yValue( )  
nodejs function get_yValue( )  
php function get_yValue( )  
cpp double get_yValue( )  
m -(double) yValue  
pas function get_yValue( ): double  
vb function get_yValue( ) As Double  
cs double get_yValue( )  
java double get_yValue( )  
py def get_yValue( )
```

### 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()**  
**gyro→zValue()gyro.get\_zValue()****YGyro**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

```
js function get_zValue( )  
node.js function get_zValue( )  
php function get_zValue( )  
cpp double get_zValue( )  
m -(double) zValue  
pas function get_zValue( ): double  
vb function get_zValue( ) As Double  
cs double get_zValue( )  
java double get_zValue( )  
py def get_zValue( )
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_ZVALUE\_INVALID.

## gyro→isOnline()gyro.isOnline()

YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

```
js function isOnline( )
node.js function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

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()

YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function loadCalibrationPoints( var rawValues: TDoubleArray,
                                      var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( )
cs    int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
py    def loadCalibrationPoints( rawValues, refValues)
cmd   YGyro target loadCalibrationPoints rawValues refValues

```

### 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()

## YGyro

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextGyro()</b>
nodejs	function <b>nextGyro()</b>
php	function <b>nextGyro()</b>
cpp	YGyro * <b>nextGyro()</b>
m	-(YGyro*) <b>nextGyro</b>
pas	function <b>nextGyro()</b> : TYGyro
vb	function <b>nextGyro()</b> As YGyro
cs	YGyro <b>nextGyro()</b>
java	YGyro <b>nextGyro()</b>
py	def <b>nextGyro()</b>

**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.

```

js   function registerAnglesCallback( callback)
nodejs function registerAnglesCallback( callback)
php  function registerAnglesCallback( $callback)
cpp   int registerAnglesCallback( YAnglesCallback callback)
m    -(int) registerAnglesCallback : (YAnglesCallback) callback
pas   function registerAnglesCallback( callback: TYAnglesCallback): LongInt
vb    function registerAnglesCallback( ) As Integer
cs   int registerAnglesCallback( YAnglesCallback callback)
java  int registerAnglesCallback( YAnglesCallback callback)
py    def registerAnglesCallback( callback)

```

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 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.

```
js function registerQuaternionCallback( callback)
nodejs function registerQuaternionCallback( callback)
php function registerQuaternionCallback( $callback)
cpp int registerQuaternionCallback( YQuatCallback callback)
m -(int) registerQuaternionCallback : (YQuatCallback) callback
pas function registerQuaternionCallback( callback: TYQuatCallback): LongInt
vb function registerQuaternionCallback( ) As Integer
cs int registerQuaternionCallback( YQuatCallback callback)
java int registerQuaternionCallback( YQuatCallback callback)
py def registerQuaternionCallback( callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appellé trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter cinq arguments: l'objet YGyro du module qui a tourné, et les valeurs des quatre composantes w, x, y et z du quaternion (nombres à virgules).

## gyro→registerTimedReportCallback() gyro.registerTimedReportCallback()

YGyro

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

js	function registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YGyroTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YGyroTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYGyroTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <b>callback</b> )

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.

## gyro→registerValueCallback() gyro.registerValueCallback()

YGyro

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YGyroValueCallback callback)
m -(int) registerValueCallback : (YGyroValueCallback) callback
pas function registerValueCallback( callback: TYGyroValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## gyro→set\_highestValue() gyro→setHighestValue()gyro.set\_highestValue()

YGYro

Modifie la mémoire de valeur maximale observée.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YGYro target set_highestValue newval
```

### Paramètres :

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## gyro→set\_logFrequency() gyro→setLogFrequency()gyro.set\_logFrequency()

YGyro

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
js function set_logFrequency( newval)
node.js function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YGyro target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

### Paramètres :

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## gyro→set\_logicalName() gyro→setLogicalName()gyro.set\_logicalName()

YGyro

Modifie le nom logique du gyroscope.

js	function <b>set_logicalName( newval)</b>
nodejs	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	<b>YGyro target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

### Paramètres :

**newval** une chaîne de caractères représentant le nom logique 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()  
gyro→setLowestValue()gyro.set\_lowestValue()****YGyro**

Modifie la mémoire de valeur minimale observée.

```
js function set_lowestValue( newval)
node.js function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YGyro target 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.

<code>js</code>	<code>function set_reportFrequency( newval)</code>
<code>node.js</code>	<code>function set_reportFrequency( newval)</code>
<code>php</code>	<code>function set_reportFrequency( \$newval)</code>
<code>cpp</code>	<code>int set_reportFrequency( const string&amp; newval)</code>
<code>m</code>	<code>-(int) setReportFrequency : (NSString*) newval</code>
<code>pas</code>	<code>function set_reportFrequency( newval: string): integer</code>
<code>vb</code>	<code>function set_reportFrequency( ByVal newval As String) As Integer</code>
<code>cs</code>	<code>int set_reportFrequency( string newval)</code>
<code>java</code>	<code>int set_reportFrequency( String newval)</code>
<code>py</code>	<code>def set_reportFrequency( newval)</code>
<code>cmd</code>	<code>YGYro target set_reportFrequency newval</code>

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() gyro→setResolution()gyro.set\_resolution()

YGyro

Modifie la résolution des valeurs physique mesurées.

```
js function set_resolution( newval)
node.js function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YGyro target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

### Paramètres :

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set(userData)****YGyro****gyro→setUserData()gyro.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

js	function <b>set(userData)</b>
nodejs	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	- <b>(void) set(userData : (void*) data)</b>
pas	procedure <b>set(userData Tobject data)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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()****YGyro**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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;
php require_once('yocto_hubport.php');
cpp #include "yocto_hubport.h"
m #import "yocto_hubport.h"
pas uses yocto_hubport;
vb yocto_hubport.vb
cs yocto_hubport.cs
java import com.yoctopuce.YoctoAPI.YHubPort;
py from yocto_hubport import *

```

### Fonction globales

#### yFindHubPort(func)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

#### yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

### Méthodes des objets YHubPort

#### hubport→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format TYPE ( NAME )=SERIAL.FUNCTIONID.

#### hubport→get\_advertisedValue()

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

#### hubport→get\_baudRate()

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

#### hubport→get\_enabled()

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

#### hubport→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

#### hubport→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

#### hubport→get\_friendlyName()

Retourne un identifiant global du port de Yocto-hub au format NOM\_MODULE.NOM\_FONCTION.

#### hubport→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### hubport→get\_functionId()

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

#### hubport→get\_hardwareId()

Retourne l'identifiant matériel unique du port de Yocto-hub au format SERIAL.FUNCTIONID.

#### hubport→get\_logicalName()

Retourne le nom logique du port de Yocto-hub.

**hubport→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**hubport→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**hubport→get\_portState()**

Retourne l'état actuel du port de Yocto-hub.

**hubport→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**hubport→isOnline()**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

**hubport→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

**hubport→load(msValidity)**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

**hubport→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

**hubport→nextHubPort()**

Continue l'énumération des port de Yocto-hub commencée à l'aide de yFirstHubPort( ).

**hubport→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**hubport→set\_enabled(newval)**

Modifie le mode d'activation du port du Yocto-hub.

**hubport→set\_logicalName(newval)**

Modifie le nom logique du port de Yocto-hub.

**hubport→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**hubport→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YHubPort.FindHubPort() yFindHubPort()YHubPort.FindHubPort()

**YHubPort**

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

js	function <b>yFindHubPort( func)</b>
node.js	function <b>FindHubPort( func)</b>
php	function <b>yFindHubPort( \$func)</b>
cpp	YHubPort* <b>yFindHubPort( const string&amp; func)</b>
m	YHubPort* <b>yFindHubPort( NSString* func)</b>
pas	function <b>yFindHubPort( func: string): TYHubPort</b>
vb	function <b>yFindHubPort( ByVal func As String) As YHubPort</b>
cs	YHubPort <b>FindHubPort( string func)</b>
java	YHubPort <b>FindHubPort( String func)</b>
py	def <b>FindHubPort( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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() YHubPort.FirstHubPort()**

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

js	function <b>yFirstHubPort( )</b>
node.js	function <b>FirstHubPort( )</b>
php	function <b>yFirstHubPort( )</b>
cpp	YHubPort* <b>yFirstHubPort( )</b>
m	YHubPort* <b>yFirstHubPort( )</b>
pas	function <b>yFirstHubPort( ): TYHubPort</b>
vb	function <b>yFirstHubPort( ) As YHubPort</b>
cs	YHubPort <b>FirstHubPort( )</b>
java	YHubPort <b>FirstHubPort( )</b>
py	def <b>FirstHubPort( )</b>

Utiliser la fonction `YHubPort.nextHubPort( )` pour itérer sur les autres port de Yocto-hub.

**Retourne :**

un pointeur sur un objet `YHubPort`, correspondant à le premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

**hubport→describe()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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YHubPort target 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.

js	function <b>get_baudRate()</b>
nodejs	function <b>get_baudRate()</b>
php	function <b>get_baudRate()</b>
cpp	int <b>get_baudRate()</b>
m	-(int) baudRate
pas	function <b>get_baudRate()</b> : LongInt
vb	function <b>get_baudRate()</b> As Integer
cs	int <b>get_baudRate()</b>
java	int <b>get_baudRate()</b>
py	def <b>get_baudRate()</b>
cmd	YHubPort <b>target get_baudRate</b>

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.

js	function <b>get_enabled()</b>
node.js	function <b>get_enabled()</b>
php	function <b>get_enabled()</b>
cpp	Y_ENABLED_enum <b>get_enabled()</b>
m	-(Y_ENABLED_enum) <b>enabled</b>
pas	function <b>get_enabled()</b> : Integer
vb	function <b>get_enabled()</b> As Integer
cs	int <b>get_enabled()</b>
java	int <b>get_enabled()</b>
py	def <b>get_enabled()</b>
cmd	YHubPort <b>target get_enabled</b>

**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.

```
js function getErrorMessage( )  
nodejs function getErrorMessage( )  
php function getErrorMessage( )  
cpp string getErrorMessage( )  
m -(NSString*) errorMessage  
pas function getErrorMessage( ): string  
vb function getErrorMessage( ) As String  
cs string getErrorMessage( )  
java String getErrorMessage( )  
py def getErrorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

```
js function get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

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.

js	function <b>get_functionDescriptor()</b>
nodejs	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) <b>functionDescriptor</b>
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**hubport→get\_functionId()****YHubPort****hubport→functionId()hubport.get\_functionId()**

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def get_functionId( )</b>

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.

js	function <b>get.hardwareId()</b> {
node.js	function <b>get.hardwareId()</b> {
php	function <b>get.hardwareId()</b> {
cpp	string <b>get.hardwareId()</b> {
m	-(NSString*) hardwareId;
vb	function <b>get.hardwareId()</b> As String
cs	string <b>get.hardwareId()</b> {
java	String <b>get.hardwareId()</b> {
py	def <b>get.hardwareId()</b> {

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.

js	function <b>get_logicalName( )</b>
nodejs	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	YHubPort <b>target get_logicalName</b>

**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()**  
**hubport→module()hubport.get\_module()****YHubPort**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module()
node.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`

**hubport→get\_module\_async()**  
**hubport→module\_async()****YHubPort**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

**hubport→get\_portState()** **YHubPort**  
**hubport→portState()hubport.get\_portState()**

Retourne l'état actuel du port de Yocto-hub.

js	function <b>get_portState()</b>
node.js	function <b>get_portState()</b>
php	function <b>get_portState()</b>
cpp	Y_PORTSTATE_enum <b>get_portState()</b>
m	-(Y_PORTSTATE_enum) portState
pas	function <b>get_portState()</b> : Integer
vb	function <b>get_portState()</b> As Integer
cs	int <b>get_portState()</b>
java	int <b>get_portState()</b>
py	def <b>get_portState()</b>
cmd	YHubPort <b>target get_portState</b>

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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()hubport.isOnline()****YHubPort**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()

## YHubPort

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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()

## YHubPort

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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.

**hubport→nextHubPort()hubport.nextHubPort()****YHubPort**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

js	function <b>nextHubPort()</b>
nodejs	function <b>nextHubPort()</b>
php	function <b>nextHubPort()</b>
cpp	<b>YHubPort * nextHubPort()</b>
m	<b>-(YHubPort*) nextHubPort</b>
pas	function <b>nextHubPort()</b> : TYHubPort
vb	function <b>nextHubPort()</b> As YHubPort
cs	<b>YHubPort nextHubPort()</b>
java	<b>YHubPort nextHubPort()</b>
py	def <b>nextHubPort()</b>

**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.

js	<code>function registerValueCallback( callback)</code>
node.js	<code>function registerValueCallback( callback)</code>
php	<code>function registerValueCallback( \$callback)</code>
cpp	<code>int registerValueCallback( YHubPortValueCallback callback)</code>
m	<code>-(int) registerValueCallback : (YHubPortValueCallback) callback</code>
pas	<code>function registerValueCallback( callback: TYHubPortValueCallback): LongInt</code>
vb	<code>function registerValueCallback( ) As Integer</code>
cs	<code>int registerValueCallback( ValueCallback callback)</code>
java	<code>int registerValueCallback( UpdateCallback callback)</code>
py	<code>def registerValueCallback( callback)</code>

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## hubport→set\_enabled() hubport→setEnabled() hubport.set\_enabled()

YHubPort

Modifie le mode d'activation du port du Yocto-hub.

```
js function set_enabled( newval)
node.js function set_enabled( newval)
php function set_enabled( $newval)
cpp int set_enabled( Y_ENABLED_enum newval)
m -(int) setEnabled : (Y_ENABLED_enum) newval
pas function set_enabled( newval: Integer): integer
vb function set_enabled( ByVal newval As Integer) As Integer
cs int set_enabled( int newval)
java int set_enabled( int newval)
py def set_enabled( newval)
cmd YHubPort target 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.

js	function <b>set_logicalName( newval)</b>
node.js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	YHubPort <b>target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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).

```
js function set(userData( data)
nodejs function set(userData( data)
php function set(userData( $data)
cpp void set(userData( void* data)
m -(void) setUserData : (void*) data
pas procedure set(userData( data: Tobject)
vb procedure set(userData( ByVal data As Object)
cs void set(userData( object data)
java void set(userData( Object data)
py def set(userData( data)
```

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()

YHubPort

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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 extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_humidity.js'></script>
nodejs var yoctolib = require('yoctolib');
var YHumidity = yoctolib.YHumidity;
require_once('yocto_humidity.php');
#include "yocto_humidity.h"
m #import "yocto_humidity.h"
pas uses yocto_humidity;
vb yocto_humidity.vb
cs yocto_humidity.cs
java import com.yoctopuce.YoctoAPI.YHumidity;
py from yocto_humidity import *

```

### Fonction globales

#### yFindHumidity(func)

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

#### yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

### Méthodes des objets YHumidity

#### humidity→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### humidity→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### humidity→get\_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

#### humidity→get\_currentRawValue()

Retourne la valeur brute rentrée par le capteur (sans arrondi ni calibration).

#### humidity→get\_currentValue()

Retourne la mesure actuelle de l'humidité.

#### humidity→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### humidity→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### humidity→get\_friendlyName()

Retourne un identifiant global du capteur d'humidité au format NOM\_MODULE . NOM\_FONCTION.

#### humidity→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### humidity→get\_functionId()

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

#### humidity→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.
<b>humidity→get_highestValue()</b>
Retourne la valeur maximale observée pour l'humidité.
<b>humidity→get_logFrequency()</b>
Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>humidity→get_logicalName()</b>
Retourne le nom logique du capteur d'humidité.
<b>humidity→get_lowestValue()</b>
Retourne la valeur minimale observée pour l'humidité.
<b>humidity→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>humidity→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>humidity→get_recordedData(startTime, endTime)</b>
Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>humidity→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>humidity→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>humidity→get_unit()</b>
Retourne l'unité dans laquelle l'humidité est exprimée.
<b>humidity→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>humidity→isOnline()</b>
Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.
<b>humidity→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.
<b>humidity→load(msValidity)</b>
Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.
<b>humidity→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>humidity→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.
<b>humidity→nextHumidity()</b>
Continue l'énumération des capteurs d'humidité commencée à l'aide de yFirstHumidity( ).
<b>humidity→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>humidity→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>humidity→set_highestValue(newval)</b>
Modifie la mémoire de valeur maximale observée pour l'humidité.
<b>humidity→set_logFrequency(newval)</b>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**humidity→set\_logicalName(newval)**

Modifie le nom logique du capteur d'humidité.

**humidity→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour l'humidité.

**humidity→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**humidity→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**humidity→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**humidity→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YHumidity.FindHumidity()****YHumidity****yFindHumidity() YHumidity.FindHumidity()**

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

js	function <b>yFindHumidity( func)</b>
node.js	function <b>FindHumidity( func)</b>
php	function <b>yFindHumidity( \$func)</b>
cpp	YHumidity* <b>yFindHumidity( const string&amp; func)</b>
m	YHumidity* <b>yFindHumidity( NSString* func)</b>
pas	function <b>yFindHumidity( func: string): TYHumidity</b>
vb	function <b>yFindHumidity( ByVal func As String) As YHumidity</b>
cs	YHumidity <b>FindHumidity( string func)</b>
java	YHumidity <b>FindHumidity( String func)</b>
py	def <b>FindHumidity( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur d'humidité soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnline()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

**Retourne :**

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

**YHumidity.FirstHumidity()****YHumidity****yFirstHumidity() YHumidity.FirstHumidity()**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

js	function <b>yFirstHumidity()</b>
node.js	function <b>FirstHumidity()</b>
php	function <b>yFirstHumidity()</b>
cpp	YHumidity* <b>yFirstHumidity()</b>
m	YHumidity* <b>yFirstHumidity()</b>
pas	function <b>yFirstHumidity()</b> : TYHumidity
vb	function <b>yFirstHumidity()</b> As YHumidity
cs	YHumidity <b>FirstHumidity()</b>
java	YHumidity <b>FirstHumidity()</b>
py	def <b>FirstHumidity()</b>

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

**Retourne :**

un pointeur sur un objet `YHumidity`, correspondant à le premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

## humidity→calibrateFromPoints() humidity.calibrateFromPoints()

YHumidity

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)

m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints()
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YHumidity target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 d'humidité (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**humidity→get\_advertisedValue()**  
**humidity→advertisedValue()**  
**humidity.get\_advertisedValue()****YHumidity**

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YHumidity <b>target get_advertisedValue</b>

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur d'humidité (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne **Y\_ADVERTISEDVALUE\_INVALID**.

**humidity→get\_currentRawValue()****YHumidity****humidity→currentRawValue()****humidity.get\_currentRawValue()**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
js function get_currentRawValue( )  
nodejs function get_currentRawValue( )  
php function get_currentRawValue( )  
cpp double get_currentRawValue( )  
m -(double) currentRawValue  
pas function get_currentRawValue( ): double  
vb function get_currentRawValue( ) As Double  
cs double get_currentRawValue( )  
java double get_currentRawValue( )  
py def get_currentRawValue( )  
cmd YHumidity target 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.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YHumidity target 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.

```
js function getErrorMessage( )
nodejs function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ): string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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.

js	function get_errorType( )
node.js	function get_errorType( )
php	function get_errorType( )
cpp	YRETCODE get_errorType( )
pas	function get_errorType( ): YRETCODE
vb	function get_errorType( ) As YRETCODE
cs	YRETCODE get_errorType( )
java	int get_errorType( )
py	def get_errorType( )

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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

**humidity→get\_functionId()****YHumidity****humidity→functionId()humidity.get\_functionId()**

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

js	function <b>get_hardwareId()</b>
nodejs	function <b>get_hardwareId()</b>
php	function <b>get_hardwareId()</b>
cpp	string <b>get_hardwareId()</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId()</b> As String
cs	string <b>get_hardwareId()</b>
java	String <b>get_hardwareId()</b>
py	def <b>get_hardwareId()</b>

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.

```
js function get_highestValue( )  
nodejs function get_highestValue( )  
php function get_highestValue( )  
cpp double get_highestValue( )  
m -(double) highestValue  
pas function get_highestValue( ): double  
vb function get_highestValue( ) As Double  
cs double get_highestValue( )  
java double get_highestValue( )  
py def get_highestValue( )  
cmd YHumidity target 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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YHumidity target get_logFrequency
```

**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.

```
js function get_logicalName( )  
node.js function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YHumidity target 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.

js	function <b>get_lowestValue( )</b>
nodejs	function <b>get_lowestValue( )</b>
php	function <b>get_lowestValue( )</b>
cpp	double <b>get_lowestValue( )</b>
m	-(double) lowestValue
pas	function <b>get_lowestValue( ): double</b>
vb	function <b>get_lowestValue( ) As Double</b>
cs	double <b>get_lowestValue( )</b>
java	double <b>get_lowestValue( )</b>
py	def <b>get_lowestValue( )</b>
cmd	<b>YHumidity target get_lowestValue</b>

**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.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As <code>YModule</code>
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YHumidity**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

## 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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
           : (s64) endTime
pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd  YHumidity target get_recordedData startTime endTime

```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

### Paramètres :

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

### Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**humidity→get\_reportFrequency()****YHumidity****humidity→reportFrequency()****humidity.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YHumidity target get_reportFrequency
```

**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.

```
js function get_resolution( )
node.js function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YHumidity target 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.

```
js function get_unit( )
nodejs function get_unit( )
php function get_unit( )
cpp string get_unit( )
m -(NSString*) unit
pas function get_unit( ): string
vb function get_unit( ) As String
cs string get_unit( )
java String get_unit( )
py def get_unit( )
cmd YHumidity target 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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData)  
m -(void*) userData  
pas function get(userData): Object  
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.

## humidity→isOnline()**humidity.isOnline()**

## YHumidity

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

```
js function isOnline( )
node.js function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

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()

YHumidity

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## humidity→loadCalibrationPoints() humidity.loadCalibrationPoints()

YHumidity

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function loadCalibrationPoints( var rawValues: TDoubleArray,
                                      var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( )
cs    int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
py    def loadCalibrationPoints( rawValues, refValues)
cmd   YHumidity target loadCalibrationPoints rawValues refValues

```

### 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()

## YHumidity

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
js   function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextHumidity()</b>
nodejs	function <b>nextHumidity()</b>
php	function <b>nextHumidity()</b>
cpp	YHumidity * <b>nextHumidity()</b>
m	-(YHumidity*) <b>nextHumidity</b>
pas	function <b>nextHumidity()</b> : TYHumidity
vb	function <b>nextHumidity()</b> As YHumidity
cs	YHumidity <b>nextHumidity()</b>
java	YHumidity <b>nextHumidity()</b>
py	def <b>nextHumidity()</b>

**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.

js	function registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YHumidityTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YHumidityTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYHumidityTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

## humidity→registerValueCallback() humidity.registerValueCallback()

YHumidity

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YHumidityValueCallback callback)
m -(int) registerValueCallback : (YHumidityValueCallback) callback
pas function registerValueCallback( callback: TYHumidityValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

<b>js</b>	function <b>set_highestValue( newval)</b>
<b>nodejs</b>	function <b>set_highestValue( newval)</b>
<b>php</b>	function <b>set_highestValue( \$newval)</b>
<b>cpp</b>	int <b>set_highestValue( double newval)</b>
<b>m</b>	-(int) setHighestValue : (double) <b>newval</b>
<b>pas</b>	function <b>set_highestValue( newval: double): integer</b>
<b>vb</b>	function <b>set_highestValue( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_highestValue( double newval)</b>
<b>java</b>	int <b>set_highestValue( double newval)</b>
<b>py</b>	def <b>set_highestValue( newval)</b>
<b>cmd</b>	YHumidity <b>target set_highestValue newval</b>

**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.

```
js function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YHumidity target 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.

js	function <b>set_logicalName( newval)</b>
node.js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	YHumidity <b>target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur d'humidité.

**Retourne :**

YAPI\_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_lowestValue()****YHumidity****humidity→setLowestValue()****humidity.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée pour l'humidité.

```
js function set_lowestValue( newval)
nodejs function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YHumidity target set_lowestValue newval
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée pour l'humidité

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## humidity→set\_reportFrequency() humidity→setReportFrequency() humidity.set\_reportFrequency()

YHumidity

Modifie la fréquence de notification périodique des valeurs mesurées.

<code>js</code>	<code>function set_reportFrequency( newval)</code>
<code>node.js</code>	<code>function set_reportFrequency( newval)</code>
<code>php</code>	<code>function set_reportFrequency( \$newval)</code>
<code>cpp</code>	<code>int set_reportFrequency( const string&amp; newval)</code>
<code>m</code>	<code>-(int) setReportFrequency : (NSString*) newval</code>
<code>pas</code>	<code>function set_reportFrequency( newval: string): integer</code>
<code>vb</code>	<code>function set_reportFrequency( ByVal newval As String) As Integer</code>
<code>cs</code>	<code>int set_reportFrequency( string newval)</code>
<code>java</code>	<code>int set_reportFrequency( String newval)</code>
<code>py</code>	<code>def set_reportFrequency( newval)</code>
<code>cmd</code>	<code>YHumidity target set_reportFrequency newval</code>

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.

```
js function set_resolution( newval)
node.js function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YHumidity target 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).

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	- <b>(void) set(userData : (void*) data)</b>
pas	procedure <b>set(userData Tobject data)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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()

YHumidity

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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.
<b>led-&gt;get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>led-&gt;get_power()</b>
Retourne l'état courant de la led.
<b>led-&gt;get_userData()</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>led-&gt;isOnline()</b>
Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.
<b>led-&gt;isOnline_async(callback, context)</b>
Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.
<b>led-&gt;load(msValidity)</b>
Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.
<b>led-&gt;load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.
<b>led-&gt;nextLed()</b>
Continue l'énumération des leds commencée à l'aide de yFirstLed( ).
<b>led-&gt;registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>led-&gt;set_blinking(newval)</b>
Modifie le mode de signalisation de la led.
<b>led-&gt;set_logicalName(newval)</b>
Modifie le nom logique de la led.
<b>led-&gt;set_luminosity(newval)</b>
Modifie l'intensité lumineuse de la led (en pour cent).
<b>led-&gt;set_power(newval)</b>
Modifie l'état courant de la led.
<b>led-&gt;set_userData(data)</b>
Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).
<b>led-&gt;wait_async(callback, context)</b>
Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YLed.FindLed() yFindLed()YLed.FindLed()

YLed

Permet de retrouver une led d'après un identifiant donné.

js	function <b>yFindLed( func)</b>
node.js	function <b>FindLed( func)</b>
php	function <b>yFindLed( \$func)</b>
cpp	YLed* <b>yFindLed( const string&amp; func)</b>
m	YLed* <b>yFindLed( NSString* func)</b>
pas	function <b>yFindLed( func: string): TYLed</b>
vb	function <b>yFindLed( ByVal func As String) As YLed</b>
cs	YLed <b>FindLed( string func)</b>
java	YLed <b>FindLed( String func)</b>
py	def <b>FindLed( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la 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() YLed.FirstLed()**

Commence l'énumération des leds accessibles par la librairie.

js	function <b>yFirstLed( )</b>
node.js	function <b>FirstLed( )</b>
php	function <b>yFirstLed( )</b>
cpp	<b>YLed*</b> <b>yFirstLed( )</b>
m	<b>YLed*</b> <b>yFirstLed( )</b>
pas	function <b>yFirstLed( )</b> : TYLed
vb	function <b>yFirstLed( )</b> As <b>YLed</b>
cs	<b>YLed FirstLed( )</b>
java	<b>YLed FirstLed( )</b>
py	def <b>FirstLed( )</b>

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**led→get\_advertisedValue()**

YLed

**led→advertisedValue()led.get\_advertisedValue()**

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
node.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 YLed target 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() led->blinking()led.get\_blinking()

YLed

Retourne le mode de signalisation de la led.

```
js function get_blinking( )
nodejs function get_blinking( )
php function get_blinking( )
cpp Y_BLINKING_enum get_blinking( )
m -(Y_BLINKING_enum) blinking
pas function get_blinking( ): Integer
vb function get_blinking( ) As Integer
cs int get_blinking( )
java int get_blinking( )
py def get_blinking( )
cmd YLed target get_blinking
```

### Retourne :

une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

En cas d'erreur, déclenche une exception ou retourne Y\_BLINKING\_INVALID.

**led→get\_errorMessage()**

YLed

**led→errorMessage()led.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

```
js function get_errorMessage( )  
node.js function get_errorMessage( )  
php function get_errorMessage( )  
cpp string get_errorMessage( )  
m -(NSString*) errorMessage  
pas function get_errorMessage( ): string  
vb function get_errorMessage( ) As String  
cs string get_errorMessage( )  
java String get_errorMessage( )  
py def get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la led.

**led->get\_errorType()****YLed****led->errorType()|led.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

js	function <b>get_errorType( )</b>
node.js	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	YRETCODE <b>get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	YRETCODE <b>get_errorType( )</b>
java	int <b>get_errorType( )</b>
py	def <b>get_errorType( )</b>

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()**  
**led→friendlyName()led.get\_friendlyName()**

YLed

Retourne un identifiant global de la led au format NOM\_MODULE . NOM\_FONCTION.

js	function get_friendlyName( )
node.js	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

Le chaîne renvoyée 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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**led→get\_functionId()  
led→functionId()led.get\_functionId()**

YLed

Retourne l'identifiant matériel de la led, sans référence au module.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

js	function <b>get.hardwareId( )</b>
nodejs	function <b>get.hardwareId( )</b>
php	function <b>get.hardwareId( )</b>
cpp	string <b>get.hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get.hardwareId( ) As String</b>
cs	string <b>get.hardwareId( )</b>
java	String <b>get.hardwareId( )</b>
py	def <b>get.hardwareId( )</b>

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()**  
**led→logicalName()led.get\_logicalName()**

YLed

Retourne le nom logique de la led.

```
js function get_logicalName( )
node.js function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YLed target 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.

js	function <b>get_luminosity()</b>
nodejs	function <b>get_luminosity()</b>
php	function <b>get_luminosity()</b>
cpp	int <b>get_luminosity()</b>
m	-(int) luminosity
pas	function <b>get_luminosity()</b> : LongInt
vb	function <b>get_luminosity()</b> As Integer
cs	int <b>get_luminosity()</b>
java	int <b>get_luminosity()</b>
py	def <b>get_luminosity()</b>
cmd	YLed target <b>get_luminosity</b>

**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()  
led→module()led.get\_module()****YLed**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module( )
node.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`

## led→get\_module\_async() led→module\_async()

YLed

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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() led->power()led.get\_power()

YLed

Retourne l'état courant de la led.

```
js function get_power( )
node.js function get_power( )
php function get_power( )
cpp Y_POWER_enum get_power( )
m -(Y_POWER_enum) power
pas function get_power( ): Integer
vb function get_power( ) As Integer
cs int get_power( )
java int get_power( )
py def get_power( )
cmd YLed target get_power
```

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

```
js function isOnline( )  
nodejs function isOnline( )  
php function isOnline( )  
cpp bool isOnline( )  
m -(BOOL) isOnline  
pas function isOnline( ): boolean  
vb function isOnline( ) As Boolean  
cs bool isOnline( )  
java boolean isOnline( )  
py def isOnline( )
```

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()

## YLed

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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()

## YLed

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
js   function load_async( msValidity, callback, context)
nodejs 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.

**led->nextLed()|led.nextLed()**

YLed

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

js	function <b>nextLed()</b>
nodejs	function <b>nextLed()</b>
php	function <b>nextLed()</b>
cpp	YLed * <b>nextLed()</b>
m	-{YLed*} <b>nextLed</b>
pas	function <b>nextLed()</b> : TYLed
vb	function <b>nextLed()</b> As YLed
cs	YLed <b>nextLed()</b>
java	YLed <b>nextLed()</b>
py	def <b>nextLed()</b>

**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.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YLedValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YLedValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYLedValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## led->set\_blinking() led->setBlinking()led.set\_blinking()

YLed

Modifie le mode de signalisation de la led.

```
js function set_blinking( newval)
node.js function set_blinking( newval)
php function set_blinking( $newval)
cpp int set_blinking( Y_BLINKING_enum newval)
m -(int) setBlinking : (Y_BLINKING_enum) newval
pas function set_blinking( newval: Integer): integer
vb function set_blinking( ByVal newval As Integer) As Integer
cs int set_blinking( int newval)
java int set_blinking( int newval)
py def set_blinking( newval)
cmd YLed target 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.

<b>js</b>	<code>function set_logicalName( newval)</code>
<b>node.js</b>	<code>function set_logicalName( newval)</code>
<b>php</b>	<code>function set_logicalName( \$newval)</code>
<b>cpp</b>	<code>int set_logicalName( const string&amp; newval)</code>
<b>m</b>	<code>-(int) setLogicalName : (NSString*) newval</code>
<b>pas</b>	<code>function set_logicalName( newval: string): integer</code>
<b>vb</b>	<code>function set_logicalName( ByVal newval As String) As Integer</code>
<b>cs</b>	<code>int set_logicalName( string newval)</code>
<b>java</b>	<code>int set_logicalName( String newval)</code>
<b>py</b>	<code>def set_logicalName( newval)</code>
<b>cmd</b>	<code>YLed target set_logicalName newval</code>

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).

```
js function set_luminosity( newval)
node.js function set_luminosity( newval)
php function set_luminosity( $newval)
cpp int set_luminosity( int newval)
m -(int) setLuminosity : (int) newval
pas function set_luminosity( newval: LongInt): integer
vb function set_luminosity( ByVal newval As Integer) As Integer
cs int set_luminosity( int newval)
java int set_luminosity( int newval)
py def set_luminosity( newval)
cmd YLed target 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.

```
js function set_power( newval)
nodejs function set_power( newval)
php function set_power( $newval)
cpp int set_power( Y_POWER_enum newval)
m -(int) setPower : (Y_POWER_enum) newval
pas function set_power( newval: Integer): integer
vb function set_power( ByVal newval As Integer) As Integer
cs int set_power( int newval)
java int set_power( int newval)
py def set_power( newval)
cmd YLed target set_power newval
```

### 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

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
js function set(userData) data
nodejs function set(userData) data
php function set(userData) $data
cpp void set(userData) void* data
m -(void) setUserData : (void*) data
pas procedure set(userData) data: Tobject
vb procedure set(userData) ByVal data As Object
cs void set(userData) object data
java void set(userData) Object data
py def set(userData) data
```

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()

## YLed

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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 extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_lightsensor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YLightSensor = yoctolib.YLightSensor;
php require_once('yocto_lightsensor.php');
cpp #include "yocto_lightsensor.h"
m #import "yocto_lightsensor.h"
pas uses yocto_lightsensor;
vb yocto_lightsensor.vb
cs yocto_lightsensor.cs
java import com.yoctopuce.YoctoAPI.YLightSensor;
py from yocto_lightsensor import *

```

### Fonction globales

#### yFindLightSensor(func)

Permet de retrouver un capteur de lumière d'après un identifiant donné.

#### yFirstLightSensor()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

### Méthodes des objets YLightSensor

#### lightsensor→calibrate(calibratedVal)

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

#### lightsensor→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### lightsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### lightsensor→get\_advertisedValue()

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

#### lightsensor→get\_currentRawValue()

Retourne la valeur brute rentrée par le capteur (sans arrondi ni calibration).

#### lightsensor→get\_currentValue()

Retourne la mesure actuelle de la lumière ambiante.

#### lightsensor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

#### lightsensor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

#### lightsensor→get\_friendlyName()

Retourne un identifiant global du capteur de lumière au format NOM\_MODULE . NOM\_FONCTION.

#### lightsensor→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### lightsensor→get\_functionId()

	Retourne l'identifiant matériel du capteur de lumière, sans référence au module.
<b>lightsensor→get_hardwareId()</b>	Retourne l'identifiant matériel unique du capteur de lumière au format SERIAL . FUNCTIONID.
<b>lightsensor→get_highestValue()</b>	Retourne la valeur maximale observée pour la lumière ambiante.
<b>lightsensor→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>lightsensor→get_logicalName()</b>	Retourne le nom logique du capteur de lumière.
<b>lightsensor→get_lowestValue()</b>	Retourne la valeur minimale observée pour la lumière ambiante.
<b>lightsensor→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>lightsensor→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>lightsensor→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>lightsensor→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>lightsensor→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>lightsensor→get_unit()</b>	Retourne l'unité dans laquelle la lumière ambiante est exprimée.
<b>lightsensor→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>lightsensor→isOnline()</b>	Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.
<b>lightsensor→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.
<b>lightsensor→load(msValidity)</b>	Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.
<b>lightsensor→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>lightsensor→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.
<b>lightsensor→nextLightSensor()</b>	Continue l'énumération des capteurs de lumière commencée à l'aide de yFirstLightSensor( ).
<b>lightsensor→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>lightsensor→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>lightsensor→set_highestValue(newval)</b>	

### 3. Reference

---

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

**[lightsensor→set\\_logFrequency\(newval\)](#)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**[lightsensor→set\\_logicalName\(newval\)](#)**

Modifie le nom logique du capteur de lumière.

**[lightsensor→set\\_lowestValue\(newval\)](#)**

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

**[lightsensor→set\\_reportFrequency\(newval\)](#)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**[lightsensor→set\\_resolution\(newval\)](#)**

Modifie la résolution des valeurs physique mesurées.

**[lightsensor→set\(userData\)](#)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**[lightsensor→wait\\_async\(callback, context\)](#)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YLightSensor.FindLightSensor()****YLightSensor****yFindLightSensor()YLightSensor.FindLightSensor()**

Permet de retrouver un capteur de lumière d'après un identifiant donné.

<code>js</code>	<code>function yFindLightSensor( func)</code>
<code>node.js</code>	<code>function FindLightSensor( func)</code>
<code>php</code>	<code>function yFindLightSensor( \$func)</code>
<code>cpp</code>	<code>YLightSensor* yFindLightSensor( const string&amp; func)</code>
<code>m</code>	<code>YLightSensor* yFindLightSensor( NSString* func)</code>
<code>pas</code>	<code>function yFindLightSensor( func: string): TYLightSensor</code>
<code>vb</code>	<code>function yFindLightSensor( ByVal func As String) As YLightSensor</code>
<code>cs</code>	<code>YLightSensor FindLightSensor( string func)</code>
<code>java</code>	<code>YLightSensor FindLightSensor( String func)</code>
<code>py</code>	<code>def FindLightSensor( 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 le capteur de lumière soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLightSensor.isOnline()` pour tester si le capteur de lumière est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

`func` une chaîne de caractères qui référence le capteur de lumière sans ambiguïté

**Retourne :**

un objet de classe `YLightSensor` qui permet ensuite de contrôler le capteur de lumière.

**YLightSensor.FirstLightSensor()****YLightSensor****yFirstLightSensor() YLightSensor.FirstLightSensor()**

Commence l'énumération des capteurs de lumière accessibles par la librairie.

js	function <b>yFirstLightSensor( )</b>
node.js	function <b>FirstLightSensor( )</b>
php	function <b>yFirstLightSensor( )</b>
cpp	YLightSensor* <b>yFirstLightSensor( )</b>
m	YLightSensor* <b>yFirstLightSensor( )</b>
pas	function <b>yFirstLightSensor( ): TYLightSensor</b>
vb	function <b>yFirstLightSensor( ) As YLightSensor</b>
cs	<b>YLightSensor FirstLightSensor( )</b>
java	<b>YLightSensor FirstLightSensor( )</b>
py	def <b>FirstLightSensor( )</b>

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

**Retourne :**

un pointeur sur un objet `YLightSensor`, correspondant à le premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

**lightsensor→calibrate()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).

```
js function calibrate( calibratedVal)
nodejs function calibrate( calibratedVal)
php function calibrate( $calibratedVal)
cpp int calibrate( double calibratedVal)
m -(int) calibrate : (double) calibratedVal
pas function calibrate( calibratedVal: double): integer
vb function calibrate( ByVal calibratedVal As Double) As Integer
cs int calibrate( double calibratedVal)
java int calibrate( double calibratedVal)
py def calibrate( calibratedVal)
cmd YLightSensor target calibrate calibratedVal
```

**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()**

**YLightSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                           vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                           refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YLightSensor target 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.

js	function <b>describe</b> ( )
node.js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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 lumière (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**lightsensor→get\_advertisedValue()**  
**lightsensor→advertisedValue()**  
**lightsensor.get\_advertisedValue()****YLightSensor**

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YLightSensor target 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).

```
js    function get_currentRawValue( )  
node.js function get_currentRawValue( )  
php   function get_currentRawValue( )  
cpp   double get_currentRawValue( )  
m     -(double) currentRawValue  
pas   function get_currentRawValue( ): double  
vb    function get_currentRawValue( ) As Double  
cs    double get_currentRawValue( )  
java  double get_currentRawValue( )  
py    def get_currentRawValue( )  
cmd   YLightSensor target get_currentRawValue
```

**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.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YLightSensor target get_currentValue
```

**Retourne :**

une valeur numérique représentant la mesure actuelle de la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**lightsensor→get\_errorMessage()**  
**lightsensor→errorMessage()**  
**lightsensor.get\_errorMessage()****YLightSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

js	function <b>get_errorMessage( )</b>
nodejs	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	String <b>get_errorMessage( )</b>
py	def <b>get_errorMessage( )</b>

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function get_friendlyName( )
node.js	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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()**  
**lightsensor→functionDescriptor()**  
**lightsensor.get\_functionDescriptor()****YLightSensor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**lightsensor→get\_functionId()****YLightSensor****lightsensor→functionId()lightsensor.get\_functionId()**

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de lumière (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**lightsensor→get\_hardwareId()**  
**lightsensor→hardwareId()**  
**lightsensor.get\_hardwareId()****YLightSensor**

Retourne l'identifiant matériel unique du capteur de lumière au format SERIAL.FUNCTIONID.

js	function get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

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.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YLightSensor target <b>get_highestValue</b>

**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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YLightSensor target get_logFrequency
```

**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.

js	function get_logicalName( )
node.js	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YLightSensor target 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.

```
js function get_lowestValue( )  
nodejs function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YLightSensor target 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.

js	function <b>get_module( )</b>
nodejs	function <b>get_module( )</b>
php	function <b>get_module( )</b>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module( ): TYModule</b>
vb	function <b>get_module( ) As YModule</b>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**lightsensor→get\_module\_async()**  
**lightsensor→module\_async()****YLightSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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.

## **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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YLightSensor target get_recordedData startTime endTime

```

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.

**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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YLightSensor target get_reportFrequency
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne **Y\_REPORTFREQUENCY\_INVALID**.

## lightsensor→get\_resolution() lightsensor→resolution()lightsensor.get\_resolution()

YLightSensor

Retourne la résolution des valeurs mesurées.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YLightSensor target get_resolution
```

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.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YLightSensor target get_unit</b>

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	<b>def isOnline()</b>

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()

## YLightSensor

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **lightsensor→loadCalibrationPoints()**

**YLightSensor**

### **lightsensor.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)

cmd  YLightSensor target loadCalibrationPoints rawValues refValues

```

#### **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()

YLightSensor

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()****YLightSensor****lightsensor.nextLightSensor()**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

<code>js</code>	<code>function nextLightSensor( )</code>
<code>nodejs</code>	<code>function nextLightSensor( )</code>
<code>php</code>	<code>function nextLightSensor( )</code>
<code>cpp</code>	<code>YLightSensor * nextLightSensor( )</code>
<code>m</code>	<code>-(YLightSensor*) nextLightSensor</code>
<code>pas</code>	<code>function nextLightSensor( ): TYLightSensor</code>
<code>vb</code>	<code>function nextLightSensor( ) As YLightSensor</code>
<code>cs</code>	<code>YLightSensor nextLightSensor( )</code>
<code>java</code>	<code>YLightSensor nextLightSensor( )</code>
<code>py</code>	<code>def nextLightSensor( )</code>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YLightSensorTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YLightSensorTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYLightSensorTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

## lightsensor→registerValueCallback() lightsensor.registerValueCallback()

**YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YLightSensorValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YLightSensorValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYLightSensorValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**lightsensor→set\_highestValue()**  
**lightsensor→setHighestValue()**  
**lightsensor.set\_highestValue()**

**YLightSensor**

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YLightSensor target 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.

<b>js</b>	function <b>set_logFrequency( newval)</b>
<b>node.js</b>	function <b>set_logFrequency( newval)</b>
<b>php</b>	function <b>set_logFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_logFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogFrequency : (NSString*) newval</b>
<b>pas</b>	function <b>set_logFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logFrequency( string newval)</b>
<b>java</b>	int <b>set_logFrequency( String newval)</b>
<b>py</b>	def <b>set_logFrequency( newval)</b>
<b>cmd</b>	<b>YLightSensor target set_logFrequency newval</b>

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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YLightSensor 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 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.

js	function <b>set_lowestValue( newval)</b>
node.js	function <b>set_lowestValue( newval)</b>
php	function <b>set_lowestValue( \$newval)</b>
cpp	int <b>set_lowestValue( double newval)</b>
m	-(int) <b>setLowestValue : (double) newval</b>
pas	function <b>set_lowestValue( newval: double): integer</b>
vb	function <b>set_lowestValue( ByVal newval As Double) As Integer</b>
cs	int <b>set_lowestValue( double newval)</b>
java	int <b>set_lowestValue( double newval)</b>
py	def <b>set_lowestValue( newval)</b>
cmd	<b>YLightSensor target set_lowestValue newval</b>

**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.

```
js function set_reportFrequency( newval)
nodejs function set_reportFrequency( newval)
php function set_reportFrequency( $newval)
cpp int set_reportFrequency( const string& newval)
m -(int) setReportFrequency : (NSString*) newval
pas function set_reportFrequency( newval: string): integer
vb function set_reportFrequency( ByVal newval As String) As Integer
cs int set_reportFrequency( string newval)
java int set_reportFrequency( String newval)
py def set_reportFrequency( newval)
cmd YLightSensor target 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.

<code>js</code>	<code>function set_resolution( newval)</code>
<code>node.js</code>	<code>function set_resolution( newval)</code>
<code>php</code>	<code>function set_resolution( \$newval)</code>
<code>cpp</code>	<code>int set_resolution( double newval)</code>
<code>m</code>	<code>-(int) setResolution : (double) newval</code>
<code>pas</code>	<code>function set_resolution( newval: double): integer</code>
<code>vb</code>	<code>function set_resolution( ByVal newval As Double) As Integer</code>
<code>cs</code>	<code>int set_resolution( double newval)</code>
<code>java</code>	<code>int set_resolution( double newval)</code>
<code>py</code>	<code>def set_resolution( newval)</code>
<code>cmd</code>	<code>YLightSensor target set_resolution newval</code>

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).

```
js function set(userData) data
nodejs function set(userData) data
php function set(userData) $data
cpp void set(userData void* data)
m -(void) setUserData : (void*) data
pas procedure set(userData data: Tobject)
vb procedure set(userData( ByVal data As Object)
cs void set(userData object data)
java void set(userData( Object data)
py def set(userData( data)
```

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()

## YLightSensor

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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 extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_magnetometer.js'></script>
nodejs var yoctolib = require('yoctolib');
var YMagnetometer = yoctolib.YMagnetometer;
php require_once('yocto_magnetometer.php');
cpp #include "yocto_magnetometer.h"
m #import "yocto_magnetometer.h"
pas uses yocto_magnetometer;
vb yocto_magnetometer.vb
cs yocto_magnetometer.cs
java import com.yoctopuce.YoctoAPI.YMagnetometer;
py from yocto_magnetometer import *

```

### Fonction globales

#### **yFindMagnetometer(func)**

Permet de retrouver un magnétomètre d'après un identifiant donné.

#### **yFirstMagnetometer()**

Commence l'énumération des magnétomètres accessibles par la librairie.

### Méthodes des objets YMagnetometer

#### **magnetometer→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **magnetometer→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **magnetometer→get\_advertisedValue()**

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

#### **magnetometer→get\_currentRawValue()**

Retourne la valeur brute renournée par le capteur (sans arrondi ni calibration).

#### **magnetometer→get\_currentValue()**

Retourne la valeur actuelle du champ magnétique.

#### **magnetometer→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### **magnetometer→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

#### **magnetometer→get\_friendlyName()**

Retourne un identifiant global du magnétomètre au format NOM\_MODULE . NOM\_FONCTION.

#### **magnetometer→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **magnetometer→get\_functionId()**

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

#### **magnetometer→get\_hardwareId()**

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL . FUNCTIONID.

<b>magnetometer→get_highestValue()</b>	Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.
<b>magnetometer→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>magnetometer→get_logicalName()</b>	Retourne le nom logique du magnétomètre.
<b>magnetometer→get_lowestValue()</b>	Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.
<b>magnetometer→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>magnetometer→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>magnetometer→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>magnetometer→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>magnetometer→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>magnetometer→get_unit()</b>	Retourne l'unité dans laquelle le champ magnétique est exprimée.
<b>magnetometer→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>magnetometer→get_xValue()</b>	Retourne la composante X du champ magnétique, sous forme de nombre à virgule.
<b>magnetometer→get_yValue()</b>	Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.
<b>magnetometer→get_zValue()</b>	Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.
<b>magnetometer→isOnline()</b>	Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.
<b>magnetometer→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.
<b>magnetometer→load(msValidity)</b>	Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.
<b>magnetometer→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>magnetometer→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.
<b>magnetometer→nextMagnetometer()</b>	Continue l'énumération des magnétomètres commencée à l'aide de yFirstMagnetometer( ).
<b>magnetometer→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.

### 3. Reference

#### **magnetometer→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **magnetometer→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

#### **magnetometer→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### **magnetometer→set\_logicalName(newval)**

Modifie le nom logique du magnétomètre.

#### **magnetometer→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

#### **magnetometer→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **magnetometer→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **magnetometer→set(userData,data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **magnetometer→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## Y Magnetometer.FindMagnetometer() yFindMagnetometer() Y Magnetometer.FindMagnetometer()

**Y Magnetometer**

Permet de retrouver un magnétomètre d'après un identifiant donné.

js	function <b>yFindMagnetometer( func)</b>
node.js	function <b>FindMagnetometer( func)</b>
php	function <b>yFindMagnetometer( \$func)</b>
cpp	Y Magnetometer* <b>yFindMagnetometer( const string&amp; func)</b>
m	Y Magnetometer* <b>yFindMagnetometer( NSString* func)</b>
pas	function <b>yFindMagnetometer( func: string): TYMagnetometer</b>
vb	function <b>yFindMagnetometer( ByVal func As String) As YMagnetometer</b>
cs	Y Magnetometer <b>FindMagnetometer( string func)</b>
java	Y Magnetometer <b>FindMagnetometer( String func)</b>
py	def <b>FindMagnetometer( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le magnétomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMagnetometer.isOnLine()` pour tester si le magnétomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le magnétomètre sans ambiguïté

### Retourne :

un objet de classe `YMagnetometer` qui permet ensuite de contrôler le magnétomètre.

**Y Magnetometer.FirstMagnetometer()****Y Magnetometer****yFirstMagnetometer()****Y Magnetometer.FirstMagnetometer()**

Commence l'énumération des magnétomètres accessibles par la librairie.

```
js function yFirstMagnetometer( )  
nodejs function FirstMagnetometer( )  
php function yFirstMagnetometer( )  
cpp YMagnetometer* yFirstMagnetometer( )  
m YMagnetometer* yFirstMagnetometer( )  
pas function yFirstMagnetometer( ): TYMagnetometer  
vb function yFirstMagnetometer( ) As YMagnetometer  
cs YMagnetometer FirstMagnetometer( )  
java YMagnetometer FirstMagnetometer( )  
py def FirstMagnetometer( )
```

Utiliser la fonction `YMagnetometer.nextMagnetometer()` pour itérer sur les autres magnétomètres.

**Retourne :**

un pointeur sur un objet `YMagnetometer`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
node.js function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)

m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints()
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YMagnetometer target 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

```
js    function get_advertisedValue( )  
node.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   YMagnetometer target 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 renvoyée par le capteur (sans arrondi ni calibration).

```
js function get_currentRawValue( )
nodejs function get_currentRawValue( )
php function get_currentRawValue( )
cpp double get_currentRawValue( )
m -(double) currentRawValue
pas function get_currentRawValue( ): double
vb function get_currentRawValue( ) As Double
cs double get_currentRawValue( )
java double get_currentRawValue( )
py def get_currentRawValue( )
cmd YMagnetometer target get_currentRawValue
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**magnetometer→get\_currentValue()**  
**magnetometer→currentValue()**  
**magnetometer.get\_currentValue()**

**YMagnetometer**

Retourne la valeur actuelle du champ magnétique.

js	function <b>get_currentValue( )</b>
node.js	function <b>get_currentValue( )</b>
php	function <b>get_currentValue( )</b>
cpp	double <b>get_currentValue( )</b>
m	-(double) <b>currentValue</b>
pas	function <b>get_currentValue( )</b> : double
vb	function <b>get_currentValue( )</b> As Double
cs	double <b>get_currentValue( )</b>
java	double <b>get_currentValue( )</b>
py	def <b>get_currentValue( )</b>
cmd	YMagnetometer <b>target get_currentValue</b>

**Retourne :**

une valeur numérique représentant la valeur actuelle du champ magnétique

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTVALUE\_INVALID**.

**magnetometer→get\_errorMessage()**  
**magnetometer→errorMessage()**  
**magnetometer.get\_errorMessage()****YMagnetometer**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
js function get_errorMessage( )
nodejs function get_errorMessage( )
php function get_errorMessage( )
cpp string get_errorMessage( )
m -(NSString*) errorMessage
pas function get_errorMessage( ):string
vb function get_errorMessage( ) As String
cs string get_errorMessage( )
java String get_errorMessage( )
py def get_errorMessage( )
```

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.

js	function <b>get_errorType( )</b>
nodejs	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	<b>YRETCODE get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	<b>YRETCODE get_errorType( )</b>
java	<b>int get_errorType( )</b>
py	<b>def get_errorType( )</b>

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

**magnetometer→get\_friendlyName()**  
**magnetometer→friendlyName()**  
**magnetometer.get\_friendlyName()****YMagnetometer**

Retourne un identifiant global du magnétomètre au format NOM\_MODULE.NOM\_FONCTION.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

```
js function get_functionDescriptor( )  
nodejs function get_functionDescriptor( )  
php function get_functionDescriptor( )  
cpp YFUN_DESCR get_functionDescriptor( )  
m -(YFUN_DESCR) functionDescriptor  
pas function get_functionDescriptor( ): YFUN_DESCR  
vb function get_functionDescriptor( ) As YFUN_DESCR  
cs YFUN_DESCR get_functionDescriptor( )  
java String get_functionDescriptor( )  
py def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**magnetometer→get\_functionId()**  
**magnetometer→functionId()**  
**magnetometer.get\_functionId()****YMagnetometer**

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

js	function <b>get_functionId( )</b>
nodejs	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( )</b> As String
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

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.

js	function <b>get_highestValue( )</b>
nodejs	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YMagnetometer <b>target get_highestValue</b>

**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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YMagnetometer target get_logFrequency
```

**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.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YMagnetometer target get_logicalName
```

**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.

js	function <b>get_lowestValue( )</b>
node.js	function <b>get_lowestValue( )</b>
php	function <b>get_lowestValue( )</b>
cpp	double <b>get_lowestValue( )</b>
m	-(double) lowestValue
pas	function <b>get_lowestValue( )</b> : double
vb	function <b>get_lowestValue( )</b> As Double
cs	double <b>get_lowestValue( )</b>
java	double <b>get_lowestValue( )</b>
py	def <b>get_lowestValue( )</b>
cmd	YMagnetometer <b>target get_lowestValue</b>

**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.

<code>js</code>	<code>function get_module( )</code>
<code>nodejs</code>	<code>function get_module( )</code>
<code>php</code>	<code>function get_module( )</code>
<code>cpp</code>	<code>YModule * get_module( )</code>
<code>m</code>	<code>-(YModule*) module</code>
<code>pas</code>	<code>function get_module( ): TYModule</code>
<code>vb</code>	<code>function get_module( ) As YModule</code>
<code>cs</code>	<code>YModule get_module( )</code>
<code>java</code>	<code>YModule get_module( )</code>
<code>py</code>	<code>def get_module( )</code>

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()****YMagnetometer**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
          : (s64) endTime
pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YMagnetometer target 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()**  
**magnetometer→reportFrequency()**  
**magnetometer.get\_reportFrequency()****YMagnetometer**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YMagnetometer target get_reportFrequency
```

**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.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YMagnetometer target get_resolution
```

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.

```
js function get_unit( )
nodejs function get_unit( )
php function get_unit( )
cpp string get_unit( )
m -(NSString*) unit
pas function get_unit( ): string
vb function get_unit( ) As String
cs string get_unit( )
java String get_unit( )
py def get_unit( )
cmd YMagnetometer target 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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData) {  
m -(void*) userData  
pas function get(userData): Tobject  
vb function get(userData) As Object  
cs object get(userData) {  
java Object get(userData)  
py def get(userData)
```

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.

```
js function get_xValue( )  
nodejs function get_xValue( )  
php function get_xValue( )  
cpp double get_xValue( )  
m -(double) xValue  
pas function get_xValue( ): double  
vb function get_xValue( ) As Double  
cs double get_xValue( )  
java double get_xValue( )  
py def get_xValue( )  
cmd YMagnetometer target get_xValue
```

**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.

```
js function get_yValue( )  
node.js function get_yValue( )  
php function get_yValue( )  
cpp double get_yValue( )  
m -(double) yValue  
pas function get_yValue( ): double  
vb function get_yValue( ) As Double  
cs double get_yValue( )  
java double get_yValue( )  
py def get_yValue( )  
cmd YMagnetometer target get_yValue
```

**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.

js	function <b>get_zValue( )</b>
node.js	function <b>get_zValue( )</b>
php	function <b>get_zValue( )</b>
cpp	double <b>get_zValue( )</b>
m	-(double) zValue
pas	function <b>get_zValue( ): double</b>
vb	function <b>get_zValue( ) As Double</b>
cs	double <b>get_zValue( )</b>
java	double <b>get_zValue( )</b>
py	def <b>get_zValue( )</b>
cmd	<b>YMagnetometer target get_zValue</b>

**Retourne :**

une valeur numérique représentant la composante Z du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_ZVALUE\_INVALID**.

**magnetometer→isOnline()magnetometer.isOnline()****YMagnetometer**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	<b>def isOnline()</b>

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()****YMagnetometer**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)

cmd  YMagnetometer target loadCalibrationPoints rawValues refValues

```

### 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()

## YMagnetometer

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextMagnetometer( )</b>
nodejs	function <b>nextMagnetometer( )</b>
php	function <b>nextMagnetometer( )</b>
cpp	YMagnetometer * <b>nextMagnetometer( )</b>
m	-(YMagnetometer*) <b>nextMagnetometer</b>
pas	function <b>nextMagnetometer( )</b> : TYMagnetometer
vb	function <b>nextMagnetometer( )</b> As YMagnetometer
cs	YMagnetometer <b>nextMagnetometer( )</b>
java	YMagnetometer <b>nextMagnetometer( )</b>
py	def <b>nextMagnetometer( )</b>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YMagnetometerTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YMagnetometerTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYMagnetometerTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

## magnetometer→registerValueCallback() magnetometer.registerValueCallback()

YMagnetometer

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YMagnetometerValueCallback callback)
m -(int) registerValueCallback : (YMagnetometerValueCallback) callback
pas function registerValueCallback( callback: TYMagnetometerValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**magnetometer→set\_highestValue()**  
**magnetometer→setHighestValue()**  
**magnetometer.set\_highestValue()**

YMagnetometer

Modifie la mémoire de valeur maximale observée.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YMagnetometer target set_highestValue newval
```

**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.

<b>js</b>	function <b>set_logFrequency( newval)</b>
<b>node.js</b>	function <b>set_logFrequency( newval)</b>
<b>php</b>	function <b>set_logFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_logFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) setLogFrequency : (NSString*) <b>newval</b>
<b>pas</b>	function <b>set_logFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logFrequency( string newval)</b>
<b>java</b>	int <b>set_logFrequency( String newval)</b>
<b>py</b>	def <b>set_logFrequency( newval)</b>
<b>cmd</b>	<b>YMagnetometer target set_logFrequency newval</b>

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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YMagnetometer 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 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.

```
js   function set_lowestValue( newval)
nodejs function set_lowestValue( newval)
php  function set_lowestValue( $newval)
cpp   int set_lowestValue( double newval)
m    -(int) setLowestValue : (double) newval
pas   function set_lowestValue( newval: double): integer
vb    function set_lowestValue( ByVal newval As Double) As Integer
cs    int set_lowestValue( double newval)
java  int set_lowestValue( double newval)
py    def set_lowestValue( newval)
cmd   YMagnetometer target set_lowestValue newval
```

**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.

```
js function set_reportFrequency( newval)
nodejs function set_reportFrequency( newval)
php function set_reportFrequency( $newval)
cpp int set_reportFrequency( const string& newval)
m -(int) setReportFrequency : (NSString*) newval
pas function set_reportFrequency( newval: string): integer
vb function set_reportFrequency( ByVal newval As String) As Integer
cs int set_reportFrequency( string newval)
java int set_reportFrequency( String newval)
py def set_reportFrequency( newval)
cmd YMagnetometer target 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.

<b>js</b>	function <b>set_resolution( newval)</b>
<b>node.js</b>	function <b>set_resolution( newval)</b>
<b>php</b>	function <b>set_resolution( \$newval)</b>
<b>cpp</b>	int <b>set_resolution( double newval)</b>
<b>m</b>	-(int) setResolution : (double) <b>newval</b>
<b>pas</b>	function <b>set_resolution( newval: double): integer</b>
<b>vb</b>	function <b>set_resolution( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_resolution( double newval)</b>
<b>java</b>	int <b>set_resolution( double newval)</b>
<b>py</b>	def <b>set_resolution( newval)</b>
<b>cmd</b>	<b>Y Magnetometer target set_resolution newval</b>

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).

```
js function set(userData) {  
nodejs function setUserData( data)  
php function setUserData( $data)  
cpp void setUserData( void* data)  
m -(void) setUserData : (void*) data  
pas procedure set(userData: Tobject)  
vb procedure setUserData( ByVal data As Object)  
cs void set(userData: object data)  
java void setUserData( Object data)  
py 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

**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.

```
js   function wait_async( callback, context )
nodejs 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.

js	function <b>get_averageValue( )</b>
node.js	function <b>get_averageValue( )</b>
php	function <b>get_averageValue( )</b>
cpp	double <b>get_averageValue( )</b>
m	-(double) <b>averageValue</b>
pas	function <b>get_averageValue( )</b> : double
vb	function <b>get_averageValue( )</b> As Double
cs	double <b>get_averageValue( )</b>
java	double <b>get_averageValue( )</b>
py	<b>def get_averageValue( )</b>

**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).

```
js function get_endTimeUTC( )  
nodejs function get_endTimeUTC( )  
php function get_endTimeUTC( )  
cpp double get_endTimeUTC( )  
m -(double) endTimeUTC  
pas function get_endTimeUTC( ): double  
vb function get_endTimeUTC( ) As Double  
cs double get_endTimeUTC( )  
java double get_endTimeUTC( )  
py def 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.

```
js function get_maxValue( )  
nodejs function get_maxValue( )  
php function get_maxValue( )  
cpp double get_maxValue( )  
m -(double) maxValue  
pas function get_maxValue( ): double  
vb function get_maxValue( ) As Double  
cs double get_maxValue( )  
java double get_maxValue( )  
py def get_maxValue( )
```

**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.

```
js function get_minValue( )
node.js function get_minValue( )
php function get_minValue( )
cpp double get_minValue( )
m -(double) minValue
pas function get_minValue( ): double
vb function get_minValue( ) As Double
cs double get_minValue( )
java double get_minValue( )
py def get_minValue( )
```

**Retourne :**

un nombre décimal correspondant à la plus petite valeur observée.

**measure→getStartTimeUTC()**  
**measure→startTimeUTC()**  
**measure.getStartTimeUTC()****YMeasure**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

js	function <b>getStartTimeUTC( )</b>
nodejs	function <b>getStartTimeUTC( )</b>
php	function <b>getStartTimeUTC( )</b>
cpp	double <b>getStartTimeUTC( )</b>
m	-(double) startTimeUTC
pas	function <b>getStartTimeUTC( )</b> : double
vb	function <b>getStartTimeUTC( )</b> As Double
cs	double <b>getStartTimeUTC( )</b>
java	double <b>getStartTimeUTC( )</b>
py	def <b>getStartTimeUTC( )</b>

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclurent une fraction décimale.

**Retourne :**

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la début de la mesure.

## 3.25. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Fonction globales

#### **yFindModule(func)**

Permet de retrouver un module d'après son numéro de série ou son nom logique.

#### **yFirstModule()**

Commence l'énumération des modules accessibles par la librairie.

### Méthodes des objets YModule

#### **module→describe()**

Retourne un court texte décrivant le module.

#### **module→download(pathname)**

Télécharge le fichier choisi du module et retourne son contenu.

#### **module→functionCount()**

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

#### **module→functionId(functionIndex)**

Retourne l'identifiant matériel de la *n*ième fonction du module.

#### **module→functionName(functionIndex)**

Retourne le nom logique de la *n*ième fonction du module.

#### **module→functionValue(functionIndex)**

Retourne la valeur publiée par la *n*ième fonction du module.

#### **module→get\_beacon()**

Retourne l'état de la balise de localisation.

#### **module→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

#### **module→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

#### **module→get\_firmwareRelease()**

Retourne la version du logiciel embarqué du module.

#### **module→get\_hardwareId()**

Retourne l'identifiant unique du module.

#### **module→get\_icon2d()**

Retourne l'icône du module.
<b>module→get_lastLogs()</b>
Retourne une chaîne de caractère contenant les derniers logs du module.
<b>module→get_logicalName()</b>
Retourne le nom logique du module.
<b>module→get_luminosity()</b>
Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).
<b>module→get_persistentSettings()</b>
Retourne l'état courant des réglages persistents du module.
<b>module→get_productId()</b>
Retourne l'identifiant USB du module, préprogrammé en usine.
<b>module→get_productName()</b>
Retourne le nom commercial du module, préprogrammé en usine.
<b>module→get_productRelease()</b>
Retourne le numéro de version matériel du module, préprogrammé en usine.
<b>module→get_rebootCountdown()</b>
Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.
<b>module→get_serialNumber()</b>
Retourne le numéro de série du module, préprogrammé en usine.
<b>module→get_upTime()</b>
Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module
<b>module→get_usbBandwidth()</b>
Retourne le nombre d'interface USB utilisé par le module.
<b>module→get_usbCurrent()</b>
Retourne le courant consommé par le module sur le bus USB, en milliampères.
<b>module→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>module→isOnline()</b>
Vérifie si le module est joignable, sans déclencher d'erreur.
<b>module→isOnline_async(callback, context)</b>
Vérifie si le module est joignable, sans déclencher d'erreur.
<b>module→load(msValidity)</b>
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
<b>module→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
<b>module→nextModule()</b>
Continue l'énumération des modules commencée à l'aide de yFirstModule( ).
<b>module→reboot(secBeforeReboot)</b>
Agende un simple redémarrage du module dans un nombre donné de secondes.
<b>module→registerLogCallback(callback)</b>
todo
<b>module→revertFromFlash()</b>
Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.
<b>module→saveToFlash()</b>

### 3. Reference

---

Sauve les réglages courants dans la mémoire non volatile du module.

**module→set\_beacon(newval)**

Allume ou éteint la balise de localisation du module.

**module→set\_logicalName(newval)**

Change le nom logique du module.

**module→set\_luminosity(newval)**

Modifie la luminosité des leds informatives du module.

**module→set\_usbBandwidth(newval)**

Modifie le nombre d'interface USB utilisé par le module.

**module→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**module→triggerFirmwareUpdate(secBeforeReboot)**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

**module→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YModule.FindModule() yFindModule()YModule.FindModule()

**YModule**

Permet de retrouver un module d'après son numéro de série ou son nom logique.

js	function <b>yFindModule( func)</b>
node.js	function <b>FindModule( func)</b>
php	function <b>yFindModule( \$func)</b>
cpp	YModule* <b>yFindModule( string func)</b>
m	+ <b>(YModule*) yFindModule : (NSString*) func</b>
pas	function <b>yFindModule( func: string): TYModule</b>
vb	function <b>yFindModule( ByVal func As String) As YModule</b>
cs	YModule <b>FindModule( string func)</b>
java	YModule <b>FindModule( String func)</b>
py	def <b>FindModule( func)</b>

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()YModule.FirstModule()**

Commence l'énumération des modules accessibles par la librairie.

js	function <b>yFirstModule( )</b>
node.js	function <b>FirstModule( )</b>
php	function <b>yFirstModule( )</b>
cpp	YModule* <b>yFirstModule( )</b>
m	YModule* <b>yFirstModule( )</b>
pas	function <b>yFirstModule( ): TYModule</b>
vb	function <b>yFirstModule( ) As YModule</b>
cs	YModule <b>FirstModule( )</b>
java	YModule <b>FirstModule( )</b>
py	def <b>FirstModule( )</b>

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.

js	function <b>describe</b> ( )
node.js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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()****YModule**

Télécharge le fichier choisi du module et retourne son contenu.

```
js function download( pathname)
nodejs function download( pathname)
php function download( $pathname)
cpp string download( string pathname)
m -(NSData*) download : (NSString*) pathname
pas function download( pathname: string): TByteArray
vb function download( ) As Byte
py def download( pathname)
cmd YModule target 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.

js	function <b>functionCount( )</b>
node.js	function <b>functionCount( )</b>
php	function <b>functionCount( )</b>
cpp	int <b>functionCount( )</b>
m	-(int) <b>functionCount</b>
pas	function <b>functionCount( ):integer</b>
vb	function <b>functionCount( ) As Integer</b>
cs	int <b>functionCount( )</b>
py	def <b>functionCount( )</b>

**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.

js	function <b>functionId</b> ( <b>functionIndex</b> )
nodejs	function <b>functionId</b> ( <b>functionIndex</b> )
php	function <b>functionId</b> ( \$ <b>functionIndex</b> )
cpp	string <b>functionId</b> ( int <b>functionIndex</b> )
m	- <b>NSString*</b> <b>functionId</b> : (int) <b>functionIndex</b>
pas	function <b>functionId</b> ( <b>functionIndex</b> : integer): string
vb	function <b>functionId</b> ( <b>ByVal functionIndex As Integer</b> ) As String
cs	string <b>functionId</b> ( int <b>functionIndex</b> )
py	def <b>functionId</b> ( <b>functionIndex</b> )

**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.

js	function <b>functionName( functionIndex)</b>
node.js	function <b>functionName( functionIndex)</b>
php	function <b>functionName( \$functionIndex)</b>
cpp	string <b>functionName( int functionIndex)</b>
m	<b>-(NSString*) functionName : (int) functionIndex</b>
pas	function <b>functionName( functionIndex: integer): string</b>
vb	function <b>functionName( ByVal functionIndex As Integer) As String</b>
cs	string <b>functionName( int functionIndex)</b>
py	<b>def functionName( functionIndex)</b>

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant au nom logique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module→functionValue()module.functionValue()****YModule**

Retourne la valeur publiée par la *n*ième fonction du module.

```
js function functionValue( functionIndex)
nodejs function functionValue( functionIndex)
php function functionValue( $functionIndex)
cpp string functionValue( int functionIndex)
m -(NSString*) functionValue : (int) functionIndex
pas function functionValue( functionIndex: integer): string
vb function functionValue( ByVal functionIndex As Integer) As String
cs string functionValue( int functionIndex)
py def functionValue( functionIndex)
```

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

---

<b>module-&gt;get_beacon()</b>	<b>YModule</b>
<b>module-&gt;beacon()module.get_beacon()</b>	

---

Retourne l'état de la balise de localisation.

```
js function get_beacon( )
nodejs function get_beacon( )
php function get_beacon( )
cpp Y_BEACON_enum get_beacon( )
m -(Y_BEACON_enum) beacon
pas function get_beacon( ): Integer
vb function get_beacon( ) As Integer
cs int get_beacon( )
java int get_beacon( )
py def get_beacon( )
cmd YModule target get_beacon
```

**Retourne :**

soit Y\_BEACON\_OFF, soit Y\_BEACON\_ON, selon l'état de la balise de localisation

En cas d'erreur, déclenche une exception ou retourne Y\_BEACON\_INVALID.

**module→get\_errorMessage()** **YModule**  
**module→errorMessage()module.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
js function get_errorMessage( )  
nodejs function get_errorMessage( )  
php function get_errorMessage( )  
cpp string get_errorMessage( )  
m -(NSString*) errorMessage  
pas function get_errorMessage( ):string  
vb function get_errorMessage( ) As String  
cs string get_errorMessage( )  
java String get_errorMessage( )  
py def get_errorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

```
js function get_firmwareRelease( )  
nodejs function get_firmwareRelease( )  
php function get_firmwareRelease( )  
cpp string get_firmwareRelease( )  
m -(NSString*) firmwareRelease  
pas function get_firmwareRelease( ): string  
vb function get_firmwareRelease( ) As String  
cs string get_firmwareRelease( )  
java String get_firmwareRelease( )  
py def get_firmwareRelease( )  
cmd YModule target get_firmwareRelease
```

**Retourne :**

une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne `Y_FIRMWARERELEASE_INVALID`.

**module→get\_hardwareId()  
module→hardwareId()module.get\_hardwareId()****YModule**

Retourne l'identifiant unique du module.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	<b>String get_hardwareId( )</b>
py	<b>def get_hardwareId( )</b>

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

**Retourne :**

une chaîne de caractères identifiant la fonction

**module→get\_icon2d()**  
**module→icon2d()****YModule**

Retourne l'icône du module.

js	function <b>get_icon2d()</b>
node.js	function <b>get_icon2d()</b>
php	function <b>get_icon2d()</b>
cpp	string <b>get_icon2d()</b>
m	-(NSData*) icon2d
pas	function <b>get_icon2d()</b> : TByteArray
vb	function <b>get_icon2d()</b> As Byte
py	def <b>get_icon2d()</b>
cmd	YModule <b>target get_icon2d</b>

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.

js	function <b>get_lastLogs()</b>
nodejs	function <b>get_lastLogs()</b>
php	function <b>get_lastLogs()</b>
cpp	string <b>get_lastLogs()</b>
m	-(NSString*) lastLogs
pas	function <b>get_lastLogs()</b> : string
vb	function <b>get_lastLogs()</b> As String
cs	string <b>get_lastLogs()</b>
java	String <b>get_lastLogs()</b>
py	def <b>get_lastLogs()</b>
cmd	<b>YModule target get_lastLogs</b>

Cette méthode retourne les derniers logs qui sont encore stockés dans le module.

**Retourne :**

une chaîne de caractère contenant les derniers logs du module.

**module→get\_logicalName()** **YModule**  
**module→logicalName()module.get\_logicalName()**

Retourne le nom logique du module.

```
js function get_logicalName( )  
node.js function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YModule target 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()  
module→luminosity()module.get\_luminosity()****YModule**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

js	function <b>get_luminosity( )</b>
nodejs	function <b>get_luminosity( )</b>
php	function <b>get_luminosity( )</b>
cpp	int <b>get_luminosity( )</b>
m	-(int) <b>luminosity</b>
pas	function <b>get_luminosity( )</b> : LongInt
vb	function <b>get_luminosity( )</b> As Integer
cs	int <b>get_luminosity( )</b>
java	int <b>get_luminosity( )</b>
py	def <b>get_luminosity( )</b>
cmd	<b>YModule target get_luminosity</b>

**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.

js	function get_persistentSettings( )
nodejs	function get_persistentSettings( )
php	function get_persistentSettings( )
cpp	Y_PERSISTENTSETTINGS_enum get_persistentSettings( )
m	-(Y_PERSISTENTSETTINGS_enum) persistentSettings
pas	function get_persistentSettings( ): Integer
vb	function get_persistentSettings( ) As Integer
cs	int get_persistentSettings( )
java	int get_persistentSettings( )
py	def get_persistentSettings( )
cmd	YModule target 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.

js	function <b>get_productId( )</b>
nodejs	function <b>get_productId( )</b>
php	function <b>get_productId( )</b>
cpp	int <b>get_productId( )</b>
m	-(int) productId
pas	function <b>get_productId( ): LongInt</b>
vb	function <b>get_productId( ) As Integer</b>
cs	int <b>get_productId( )</b>
java	int <b>get_productId( )</b>
py	def <b>get_productId( )</b>
cmd	<b>YModule target get_productId</b>

**Retourne :**

un entier représentant l'identifiant USB du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne **Y\_PRODUCTID\_INVALID**.

**module→get\_productName()** YModule  
**module→productName()module.get\_productName()**

Retourne le nom commercial du module, préprogrammé en usine.

```
js function get_productName( )
node.js function get_productName( )
php function get_productName( )
cpp string get_productName( )
m -(NSString*) productName
pas function get_productName( ): string
vb function get_productName( ) As String
cs string get_productName( )
java String get_productName( )
py def get_productName( )
cmd YModule target 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.

<code>js</code>	<code>function get_productRelease( )</code>
<code>node.js</code>	<code>function get_productRelease( )</code>
<code>php</code>	<code>function get_productRelease( )</code>
<code>cpp</code>	<code>int get_productRelease( )</code>
<code>m</code>	<code>-(int) productRelease</code>
<code>pas</code>	<code>function get_productRelease( ): LongInt</code>
<code>vb</code>	<code>function get_productRelease( ) As Integer</code>
<code>cs</code>	<code>int get_productRelease( )</code>
<code>java</code>	<code>int get_productRelease( )</code>
<code>py</code>	<code>def get_productRelease( )</code>
<code>cmd</code>	<code>YModule target get_productRelease</code>

**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.

js	function <b>get_rebootCountdown( )</b>
nodejs	function <b>get_rebootCountdown( )</b>
php	function <b>get_rebootCountdown( )</b>
cpp	int <b>get_rebootCountdown( )</b>
m	-(int) rebootCountdown
pas	function <b>get_rebootCountdown( )</b> : LongInt
vb	function <b>get_rebootCountdown( )</b> As Integer
cs	int <b>get_rebootCountdown( )</b>
java	int <b>get_rebootCountdown( )</b>
py	def <b>get_rebootCountdown( )</b>
cmd	YModule <b>target get_rebootCountdown</b>

**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.

```
js    function get_serialNumber( )  
nodejs function get_serialNumber( )  
php   function get_serialNumber( )  
cpp   string get_serialNumber( )  
m     -(NSString*) serialNumber  
pas   function get_serialNumber( ): string  
vb    function get_serialNumber( ) As String  
cs    string get_serialNumber( )  
java  String get_serialNumber( )  
py    def get_serialNumber( )  
cmd   YModule target get_serialNumber
```

**Retourne :**

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_SERIALNUMBER\_INVALID.

**module→get\_upTime()**  
**module→upTime()module.get\_upTime()****YModule**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

js	function <b>get_upTime( )</b>
node.js	function <b>get_upTime( )</b>
php	function <b>get_upTime( )</b>
cpp	s64 <b>get_upTime( )</b>
m	-(s64) upTime
pas	function <b>get_upTime( ): int64</b>
vb	function <b>get_upTime( ) As Long</b>
cs	long <b>get_upTime( )</b>
java	long <b>get_upTime( )</b>
py	def <b>get_upTime( )</b>
cmd	YModule <b>target get_upTime</b>

**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.

js	function <b>get_usbBandwidth()</b>
node.js	function <b>get_usbBandwidth()</b>
php	function <b>get_usbBandwidth()</b>
cpp	Y_USBBANDWIDTH_enum <b>get_usbBandwidth()</b>
m	-(Y_USBBANDWIDTH_enum) usbBandwidth
pas	function <b>get_usbBandwidth()</b> : Integer
vb	function <b>get_usbBandwidth()</b> As Integer
cs	int <b>get_usbBandwidth()</b>
java	int <b>get_usbBandwidth()</b>
py	def <b>get_usbBandwidth()</b>
cmd	YModule <b>target get_usbBandwidth</b>

**Retourne :**

soit Y\_USBBANDWIDTH\_SIMPLE, soit Y\_USBBANDWIDTH\_DOUBLE, selon le nombre d'interface USB utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_USBBANDWIDTH\_INVALID.

**module→get\_usbCurrent()****YModule****module→usbCurrent()module.get\_usbCurrent()**

Retourne le courant consommé par le module sur le bus USB, en milliampères.

**js** function **get\_usbCurrent( )****node.js** function **get\_usbCurrent( )****php** function **get\_usbCurrent( )****cpp** int **get\_usbCurrent( )****m** -(int) **usbCurrent****pas** function **get\_usbCurrent( )**: LongInt**vb** function **get\_usbCurrent( )** As Integer**cs** int **get\_usbCurrent( )****java** int **get\_usbCurrent( )****py** def **get\_usbCurrent( )****cmd** YModule target **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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()

## YModule

Vérifie si le module est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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()

## YModule

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
js   function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextModule()</b>
nodejs	function <b>nextModule()</b>
php	function <b>nextModule()</b>
cpp	<b>YModule</b> * <b>nextModule()</b>
m	-( <b>YModule</b> *) <b>nextModule</b>
pas	function <b>nextModule()</b> : TYModule
vb	function <b>nextModule()</b> As <b>YModule</b>
cs	<b>YModule</b> <b>nextModule()</b>
java	<b>YModule</b> <b>nextModule()</b>
py	def <b>nextModule()</b>

**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.

js	function reboot( secBeforeReboot)
node.js	function reboot( secBeforeReboot)
php	function reboot( \$secBeforeReboot)
cpp	int reboot( int secBeforeReboot)
m	-(int) reboot : (int) secBeforeReboot
pas	function reboot( secBeforeReboot: LongInt): LongInt
vb	function reboot( ) As Integer
cs	int reboot( int secBeforeReboot)
java	int reboot( int secBeforeReboot)
py	def reboot( secBeforeReboot)
cmd	YModule target reboot secBeforeReboot

**Paramètres :**

**secBeforeReboot** nombre de secondes avant de redémarrer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→registerLogCallback()**  
**module.registerLogCallback()****YModule**

todo

cpp	void registerLogCallback( YModuleLogCallback <b>callback</b> )
m	- (void) registerLogCallback : (YModuleLogCallback) <b>callback</b>
vb	function registerLogCallback( ByVal <b>callback</b> As YModuleLogCallback) As Integer
cs	int registerLogCallback( LogCallback <b>callback</b> )
py	def registerLogCallback( <b>callback</b> )

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**module→revertFromFlash()****YModule****module.revertFromFlash()**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

js	function revertFromFlash( )
nodejs	function revertFromFlash( )
php	function revertFromFlash( )
cpp	int revertFromFlash( )
m	-(int) revertFromFlash
pas	function revertFromFlash( ): LongInt
vb	function revertFromFlash( ) As Integer
cs	int revertFromFlash( )
java	int revertFromFlash( )
py	def revertFromFlash( )
cmd	YModule target revertFromFlash

**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.

```
js function saveToFlash( )  
nodejs function saveToFlash( )  
php function saveToFlash( )  
cpp int saveToFlash( )  
m -(int) saveToFlash  
pas function saveToFlash( ): LongInt  
vb function saveToFlash( ) As Integer  
cs int saveToFlash( )  
java int saveToFlash( )  
py def saveToFlash( )  
cmd YModule target 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.

**module→set\_beacon()**  
**module→setBeacon()module.set\_beacon()****YModule**

Allume ou éteint la balise de localisation du module.

js	function <b>set_beacon( newval)</b>
nodejs	function <b>set_beacon( newval)</b>
php	function <b>set_beacon( \$newval)</b>
cpp	int <b>set_beacon( Y_BEACON_enum newval)</b>
m	-(int) setBeacon : (Y_BEACON_enum) <b>newval</b>
pas	function <b>set_beacon( newval: Integer): integer</b>
vb	function <b>set_beacon( ByVal newval As Integer) As Integer</b>
cs	int <b>set_beacon( int newval)</b>
java	int <b>set_beacon( int newval)</b>
py	def <b>set_beacon( newval)</b>
cmd	<b>YModule target set_beacon newval</b>

**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.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YModule 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

**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.

js	function <b>set_luminosity( newval)</b>
nodejs	function <b>set_luminosity( newval)</b>
php	function <b>set_luminosity( \$newval)</b>
cpp	int <b>set_luminosity( int newval)</b>
m	-(int) setLuminosity : (int) <b>newval</b>
pas	function <b>set_luminosity( newval: LongInt): integer</b>
vb	function <b>set_luminosity( ByVal newval As Integer) As Integer</b>
cs	int <b>set_luminosity( int newval)</b>
java	int <b>set_luminosity( int newval)</b>
py	def <b>set_luminosity( newval)</b>
cmd	<b>YModule target set_luminosity newval</b>

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.

<b>js</b>	function <b>set_usbBandwidth( newval)</b>
<b>nodejs</b>	function <b>set_usbBandwidth( newval)</b>
<b>php</b>	function <b>set_usbBandwidth( \$newval)</b>
<b>cpp</b>	int <b>set_usbBandwidth( Y_USBBANDWIDTH_enum newval)</b>
<b>m</b>	-(int) <b>setUsbBandwidth : (Y_USBBANDWIDTH_enum) newval</b>
<b>pas</b>	function <b>set_usbBandwidth( newval: Integer): integer</b>
<b>vb</b>	function <b>set_usbBandwidth( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_usbBandwidth( int newval)</b>
<b>java</b>	int <b>set_usbBandwidth( int newval)</b>
<b>py</b>	def <b>set_usbBandwidth( newval)</b>
<b>cmd</b>	<b>YModule target set_usbBandwidth newval</b>

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)**  
**module→setUserData()module.set(userData)****YModule**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

js	function set(userData( data)
node.js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

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.

```
js function triggerFirmwareUpdate( secBeforeReboot)
node.js function triggerFirmwareUpdate( secBeforeReboot)
php function triggerFirmwareUpdate( $secBeforeReboot)
cpp int triggerFirmwareUpdate( int secBeforeReboot)
m -(int) triggerFirmwareUpdate : (int) secBeforeReboot
pas function triggerFirmwareUpdate( secBeforeReboot: LongInt): LongInt
vb function triggerFirmwareUpdate( ) As Integer
cs int triggerFirmwareUpdate( int secBeforeReboot)
java int triggerFirmwareUpdate( int secBeforeReboot)
py def triggerFirmwareUpdate( secBeforeReboot)
cmd YModule target triggerFirmwareUpdate secBeforeReboot
```

**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()

## YModule

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context )
nodejs 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()****YNetwork****yFindNetwork()YNetwork.FindNetwork()**

Permet de retrouver une interface réseau d'après un identifiant donné.

```
js function yFindNetwork( func)
node.js function FindNetwork( func)
php function yFindNetwork( $func)
cpp YNetwork* yFindNetwork( const string& func)
m YNetwork* yFindNetwork( NSString* func)
pas function yFindNetwork( func: string): TYNnetwork
vb function yFindNetwork( ByVal func As String) As YNetwork
cs YNetwork FindNetwork( string func)
java YNetwork FindNetwork( String func)
py def FindNetwork( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YNetwork.isOnline()` pour tester si l'interface réseau est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'interface réseau sans ambiguïté

**Retourne :**

un objet de classe `YNetwork` qui permet ensuite de contrôler l'interface réseau.

## YNetwork.FirstNetwork()

### yFirstNetwork() YNetwork.FirstNetwork()

## YNetwork

Commence l'énumération des interfaces réseau accessibles par la librairie.

js	function <b>yFirstNetwork()</b>
nodejs	function <b>FirstNetwork()</b>
php	function <b>yFirstNetwork()</b>
cpp	YNetwork* <b>yFirstNetwork()</b>
m	YNetwork* <b>yFirstNetwork()</b>
pas	function <b>yFirstNetwork()</b> : TYNetwork
vb	function <b>yFirstNetwork()</b> As YNetwork
cs	YNetwork <b>FirstNetwork()</b>
java	YNetwork <b>FirstNetwork()</b>
py	def <b>FirstNetwork()</b>

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.

```
js function callbackLogin( username, password)
nodejs function callbackLogin( username, password)
php function callbackLogin( $username, $password)
cpp int callbackLogin( string username, string password)
m -(int) callbackLogin : (NSString*) username : (NSString*) password
pas function callbackLogin( username: string, password: string): integer
vb function callbackLogin( ByVal username As String,
                           ByVal password As String) As Integer
cs int callbackLogin( string username, string password)
java int callbackLogin( String username, String password)
py def callbackLogin( username, password)
cmd YNetwork target 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.

js	function <b>describe</b> ( )
node.js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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 (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.

```
js function get_adminPassword( )
nodejs function get_adminPassword( )
php function get_adminPassword( )
cpp string get_adminPassword( )
m -(NSString*) adminPassword
pas function get_adminPassword( ): string
vb function get_adminPassword( ) As String
cs string get_adminPassword( )
java String get_adminPassword( )
py def get_adminPassword( )
cmd YNetwork target 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YNetwork <b>target get_advertisedValue</b>

**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.

js	function get_callbackCredentials( )
nodejs	function get_callbackCredentials( )
php	function get_callbackCredentials( )
cpp	string get_callbackCredentials( )
m	-(NSString*) callbackCredentials
pas	function get_callbackCredentials( ): string
vb	function get_callbackCredentials( ) As String
cs	string get_callbackCredentials( )
java	String get_callbackCredentials( )
py	def get_callbackCredentials( )
cmd	YNetwork target 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.

js	function <b>get_callbackEncoding( )</b>
node.js	function <b>get_callbackEncoding( )</b>
php	function <b>get_callbackEncoding( )</b>
cpp	Y_CALLBACKENCODING_enum <b>get_callbackEncoding( )</b>
m	-(Y_CALLBACKENCODING_enum) <b>callbackEncoding</b>
pas	function <b>get_callbackEncoding( )</b> : Integer
vb	function <b>get_callbackEncoding( )</b> As Integer
cs	int <b>get_callbackEncoding( )</b>
java	int <b>get_callbackEncoding( )</b>
py	def <b>get_callbackEncoding( )</b>
cmd	YNetwork <b>target get_callbackEncoding</b>

**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.

```
js function get_callbackMaxDelay( )  
nodejs function get_callbackMaxDelay( )  
php function get_callbackMaxDelay( )  
cpp int get_callbackMaxDelay( )  
m -(int) callbackMaxDelay  
pas function get_callbackMaxDelay( ): LongInt  
vb function get_callbackMaxDelay( ) As Integer  
cs int get_callbackMaxDelay( )  
java int get_callbackMaxDelay( )  
py def get_callbackMaxDelay( )  
cmd YNetwork target 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.

js	function <b>get_callbackMethod( )</b>
node.js	function <b>get_callbackMethod( )</b>
php	function <b>get_callbackMethod( )</b>
cpp	Y_CALLBACKMETHOD_enum <b>get_callbackMethod( )</b>
m	-(Y_CALLBACKMETHOD_enum) <b>callbackMethod</b>
pas	function <b>get_callbackMethod( )</b> : Integer
vb	function <b>get_callbackMethod( ) As Integer</b>
cs	int <b>get_callbackMethod( )</b>
java	int <b>get_callbackMethod( )</b>
py	def <b>get_callbackMethod( )</b>
cmd	YNetwork <b>target get_callbackMethod</b>

**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.

```
js function get_callbackMinDelay( )  
nodejs function get_callbackMinDelay( )  
php function get_callbackMinDelay( )  
cpp int get_callbackMinDelay( )  
m -(int) callbackMinDelay  
pas function get_callbackMinDelay( ): LongInt  
vb function get_callbackMinDelay( ) As Integer  
cs int get_callbackMinDelay( )  
java int get_callbackMinDelay( )  
py def get_callbackMinDelay( )  
cmd YNetwork target 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.

```
js function get_callbackUrl( )  
nodejs function get_callbackUrl( )  
php function get_callbackUrl( )  
cpp string get_callbackUrl( )  
m -(NSString*) callbackUrl  
pas function get_callbackUrl( ): string  
vb function get_callbackUrl( ) As String  
cs string get_callbackUrl( )  
java String get_callbackUrl( )  
py def get_callbackUrl( )  
cmd YNetwork target get_callbackUrl
```

**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).

```
js function get_discoverable( )
nodejs function get_discoverable( )
php function get_discoverable( )
cpp Y_DISCOVERABLE_enum get_discoverable( )
m -(Y_DISCOVERABLE_enum) discoverable
pas function get_discoverable( ): Integer
vb function get_discoverable( ) As Integer
cs int get_discoverable( )
java int get_discoverable( )
py def get_discoverable( )
cmd YNetwork target get_discoverable
```

**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.

js	function <b>get_errorMessage( )</b>
nodejs	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	String <b>get_errorMessage( )</b>
py	def <b>get_errorMessage( )</b>

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

```
js function get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

```
js function get_hardwareId( )  
node.js function get_hardwareId( )  
php function get_hardwareId( )  
cpp string get_hardwareId( )  
m -(NSString*) hardwareId  
vb function get_hardwareId( ) As String  
cs string get_hardwareId( )  
java String get_hardwareId( )  
py def get_hardwareId( )
```

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.

```
js function get_ipAddress( )
nodejs function get_ipAddress( )
php function get_ipAddress( )
cpp string get_ipAddress( )
m -(NSString*) ipAddress
pas function get_ipAddress( ): string
vb function get_ipAddress( ) As String
cs string get_ipAddress( )
java String get_ipAddress( )
py def get_ipAddress( )
cmd YNetwork target 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.

```
js function get_logicalName( )
node.js function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YNetwork target 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.

```
js function get_macAddress( )
nodejs function get_macAddress( )
php function get_macAddress( )
cpp string get_macAddress( )
m -(NSString*) macAddress
pas function get_macAddress( ): string
vb function get_macAddress( ) As String
cs string get_macAddress( )
java String get_macAddress( )
py def get_macAddress( )
cmd YNetwork target get_macAddress
```

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()  
network→module()network.get\_module()****YNetwork**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module( )
node.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`

**network→get\_module\_async()  
network→module\_async()****YNetwork**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

**js** function **get\_poeCurrent( )****node.js** function **get\_poeCurrent( )****php** function **get\_poeCurrent( )****cpp** int **get\_poeCurrent( )****m** -(int) **poeCurrent****pas** function **get\_poeCurrent( )**: LongInt**vb** function **get\_poeCurrent( )** As Integer**cs** int **get\_poeCurrent( )****java** int **get\_poeCurrent( )****py** def **get\_poeCurrent( )****cmd** YNetwork **target get\_poeCurrent**

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

**Retourne :**

un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne **Y\_POECURRENT\_INVALID**.

**network→get\_primaryDNS()****YNetwork****network→primaryDNS()network.get\_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

js	function <b>get_primaryDNS( )</b>
node.js	function <b>get_primaryDNS( )</b>
php	function <b>get_primaryDNS( )</b>
cpp	string <b>get_primaryDNS( )</b>
m	-(NSString*) primaryDNS
pas	function <b>get_primaryDNS( ): string</b>
vb	function <b>get_primaryDNS( ) As String</b>
cs	string <b>get_primaryDNS( )</b>
java	String <b>get_primaryDNS( )</b>
py	def <b>get_primaryDNS( )</b>
cmd	YNetwork <b>target get_primaryDNS</b>

**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.

js	function <b>get_readiness( )</b>
node.js	function <b>get_readiness( )</b>
php	function <b>get_readiness( )</b>
cpp	Y_READINESS_enum <b>get_readiness( )</b>
m	-(Y_READINESS_enum) <b>readiness</b>
pas	function <b>get_readiness( )</b> : Integer
vb	function <b>get_readiness( )</b> As Integer
cs	int <b>get_readiness( )</b>
java	int <b>get_readiness( )</b>
py	def <b>get_readiness( )</b>
cmd	<b>YNetwork target get_readiness</b>

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*).

js	function <b>get_router( )</b>
node.js	function <b>get_router( )</b>
php	function <b>get_router( )</b>
cpp	string <b>get_router( )</b>
m	-(NSString*) router
pas	function <b>get_router( )</b> : string
vb	function <b>get_router( )</b> As String
cs	string <b>get_router( )</b>
java	String <b>get_router( )</b>
py	def <b>get_router( )</b>
cmd	YNetwork <b>target get_router</b>

**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.

```
js function get_secondaryDNS( )
nodejs function get_secondaryDNS( )
php function get_secondaryDNS( )
cpp string get_secondaryDNS( )
m -(NSString*) secondaryDNS
pas function get_secondaryDNS( ): string
vb function get_secondaryDNS( ) As String
cs string get_secondaryDNS( )
java String get_secondaryDNS( )
py def get_secondaryDNS( )
cmd YNetwork target get_secondaryDNS
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_SECONDARYDNS\_INVALID.

**network→get\_subnetMask()****YNetwork****network→subnetMask()network.get\_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

```
js function get_subnetMask( )  
nodejs function get_subnetMask( )  
php function get_subnetMask( )  
cpp string get_subnetMask( )  
m -(NSString*) subnetMask  
pas function get_subnetMask( ): string  
vb function get_subnetMask( ) As String  
cs string get_subnetMask( )  
java String get_subnetMask( )  
py def get_subnetMask( )  
cmd YNetwork target get_subnetMask
```

**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).

```
js function get(userData) 
nodejs function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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.

```
js function get_userPassword( )
nodejs function get_userPassword( )
php function get_userPassword( )
cpp string get_userPassword( )
m -(NSString*) userPassword
pas function get_userPassword( ): string
vb function get_userPassword( ) As String
cs string get_userPassword( )
java String get_userPassword( )
py def get_userPassword( )
cmd YNetwork target get_userPassword
```

**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.

```
js function get_wwwWatchdogDelay( )
nodejs function get_wwwWatchdogDelay( )
php function get_wwwWatchdogDelay( )
cpp int get_wwwWatchdogDelay( )
m -(int) wwwWatchdogDelay
pas function get_wwwWatchdogDelay( ): LongInt
vb function get_wwwWatchdogDelay( ) As Integer
cs int get_wwwWatchdogDelay( )
java int get_wwwWatchdogDelay( )
py def get_wwwWatchdogDelay( )
cmd YNetwork target 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.

```
js function isOnline( )
node.js function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

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()

## YNetwork

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()

YNetwork

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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.

**network→nextNetwork()network.nextNetwork()****YNetwork**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

<code>js</code>	<code>function nextNetwork( )</code>
<code>nodejs</code>	<code>function nextNetwork( )</code>
<code>php</code>	<code>function nextNetwork( )</code>
<code>cpp</code>	<code>YNetwork * nextNetwork( )</code>
<code>m</code>	<code>-(YNetwork*) nextNetwork</code>
<code>pas</code>	<code>function nextNetwork( ): TYNetwork</code>
<code>vb</code>	<code>function nextNetwork( ) As YNetwork</code>
<code>cs</code>	<code>YNetwork nextNetwork( )</code>
<code>java</code>	<code>YNetwork nextNetwork( )</code>
<code>py</code>	<code>def nextNetwork( )</code>

**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.

```
js function ping( host)
nodejs function ping( host)
php function ping( $host)
cpp string ping( string host)
m -(NSString*) ping : (NSString*) host
pas function ping( host: string): string
vb function ping( ) As String
cs string ping( string host)
java String ping( String host)
py def ping( host)
cmd YNetwork target ping host
```

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.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YNetworkValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YNetworkValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYNetworkValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**network→set\_adminPassword()**  
**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.

```
js function set_adminPassword( newval)
nodejs function set_adminPassword( newval)
php function set_adminPassword( $newval)
cpp int set_adminPassword( const string& newval)
m -(int) setAdminPassword : (NSString*) newval
pas function set_adminPassword( newval: string): integer
vb function set_adminPassword( ByVal newval As String) As Integer
cs int set_adminPassword( string newval)
java int set_adminPassword( String newval)
py def set_adminPassword( newval)
cmd YNetwork target 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.

<b>js</b>	function <b>set_callbackCredentials( newval)</b>
<b>node.js</b>	function <b>set_callbackCredentials( newval)</b>
<b>php</b>	function <b>set_callbackCredentials( \$newval)</b>
<b>cpp</b>	int <b>set_callbackCredentials( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setCallbackCredentials : (NSString*) newval</b>
<b>pas</b>	function <b>set_callbackCredentials( newval: string): integer</b>
<b>vb</b>	function <b>set_callbackCredentials( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_callbackCredentials( string newval)</b>
<b>java</b>	int <b>set_callbackCredentials( String newval)</b>
<b>py</b>	def <b>set_callbackCredentials( newval)</b>
<b>cmd</b>	<b>YNetwork target set_callbackCredentials newval</b>

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.

```
js function set_callbackEncoding( newval)
nodejs function set_callbackEncoding( newval)
php function set_callbackEncoding( $newval)
cpp int set_callbackEncoding( Y_CALLBACKENCODING_enum newval)
m -(int) setCallbackEncoding : (Y_CALLBACKENCODING_enum) newval
pas function set_callbackEncoding( newval: Integer): integer
vb function set_callbackEncoding( ByVal newval As Integer) As Integer
cs int set_callbackEncoding( int newval)
java int set_callbackEncoding( int newval)
py def set_callbackEncoding( newval)
cmd YNetwork target 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.

js	function <b>set_callbackMaxDelay( newval)</b>
node.js	function <b>set_callbackMaxDelay( newval)</b>
php	function <b>set_callbackMaxDelay( \$newval)</b>
cpp	int <b>set_callbackMaxDelay( int newval)</b>
m	-(int) setCallbackMaxDelay : (int) <b>newval</b>
pas	function <b>set_callbackMaxDelay( newval: LongInt): integer</b>
vb	function <b>set_callbackMaxDelay( ByVal newval As Integer) As Integer</b>
cs	int <b>set_callbackMaxDelay( int newval)</b>
java	int <b>set_callbackMaxDelay( int newval)</b>
py	def <b>set_callbackMaxDelay( newval)</b>
cmd	YNetwork <b>target set_callbackMaxDelay newval</b>

#### 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.

```
js function set_callbackMethod( newval)
nodejs function set_callbackMethod( newval)
php function set_callbackMethod( $newval)
cpp int set_callbackMethod( Y_CALLBACKMETHOD_enum newval)
m -(int) setCallbackMethod : (Y_CALLBACKMETHOD_enum) newval
pas function set_callbackMethod( newval: Integer): integer
vb function set_callbackMethod( ByVal newval As Integer) As Integer
cs int set_callbackMethod( int newval)
java int set_callbackMethod( int newval)
py def set_callbackMethod( newval)
cmd YNetwork target 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.

<code>js</code>	function <b>set_callbackMinDelay( newval)</b>
<code>nodejs</code>	function <b>set_callbackMinDelay( newval)</b>
<code>php</code>	function <b>set_callbackMinDelay( \$newval)</b>
<code>cpp</code>	int <b>set_callbackMinDelay( int newval)</b>
<code>m</code>	-(int) setCallbackMinDelay : (int) <b>newval</b>
<code>pas</code>	function <b>set_callbackMinDelay( newval: LongInt): integer</b>
<code>vb</code>	function <b>set_callbackMinDelay( ByVal newval As Integer) As Integer</b>
<code>cs</code>	int <b>set_callbackMinDelay( int newval)</b>
<code>java</code>	int <b>set_callbackMinDelay( int newval)</b>
<code>py</code>	def <b>set_callbackMinDelay( newval)</b>
<code>cmd</code>	<b>YNetwork target set_callbackMinDelay newval</b>

**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.

js	function set_callbackUrl( newval)
node.js	function set_callbackUrl( newval)
php	function set_callbackUrl( \$newval)
cpp	int set_callbackUrl( const string& newval)
m	-(int) setCallbackUrl : (NSString*) newval
pas	function set_callbackUrl( newval: string): integer
vb	function set_callbackUrl( ByVal newval As String) As Integer
cs	int set_callbackUrl( string newval)
java	int set_callbackUrl( String newval)
py	def set_callbackUrl( newval)
cmd	YNetwork target 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).

<b>js</b>	function <b>set_discoverable( newval)</b>
<b>nodejs</b>	function <b>set_discoverable( newval)</b>
<b>php</b>	function <b>set_discoverable( \$newval)</b>
<b>cpp</b>	int <b>set_discoverable( Y_DISCOVERABLE_enum newval)</b>
<b>m</b>	-(int) setDiscoverable : (Y_DISCOVERABLE_enum) <b>newval</b>
<b>pas</b>	function <b>set_discoverable( newval: Integer): integer</b>
<b>vb</b>	function <b>set_discoverable( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_discoverable( int newval)</b>
<b>java</b>	int <b>set_discoverable( int newval)</b>
<b>py</b>	def <b>set_discoverable( newval)</b>
<b>cmd</b>	<b>YNetwork target set_discoverable newval</b>

#### 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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YNetwork 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 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.

<b>js</b>	function <b>set_primaryDNS( newval)</b>
<b>nodejs</b>	function <b>set_primaryDNS( newval)</b>
<b>php</b>	function <b>set_primaryDNS( \$newval)</b>
<b>cpp</b>	int <b>set_primaryDNS( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setPrimaryDNS : (NSString*) newval</b>
<b>pas</b>	function <b>set_primaryDNS( newval: string): integer</b>
<b>vb</b>	function <b>set_primaryDNS( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_primaryDNS( string newval)</b>
<b>java</b>	int <b>set_primaryDNS( String newval)</b>
<b>py</b>	def <b>set_primaryDNS( newval)</b>
<b>cmd</b>	<b>YNetwork target set_primaryDNS newval</b>

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.

```
js function set_secondaryDNS( newval)
nodejs function set_secondaryDNS( newval)
php function set_secondaryDNS( $newval)
cpp int set_secondaryDNS( const string& newval)
m -(int) setSecondaryDNS : (NSString*) newval
pas function set_secondaryDNS( newval: string): integer
vb function set_secondaryDNS( ByVal newval As String) As Integer
cs int set_secondaryDNS( string newval)
java int set_secondaryDNS( String newval)
py def set_secondaryDNS( newval)
cmd YNetwork target 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).

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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.

```
js function set_userPassword( newval)
nodejs function set_userPassword( newval)
php function set_userPassword( $newval)
cpp int set_userPassword( const string& newval)
m -(int) setUserPassword : (NSString*) newval
pas function set_userPassword( newval: string): integer
vb function set_userPassword( ByVal newval As String) As Integer
cs int set_userPassword( string newval)
java int set_userPassword( String newval)
py def set_userPassword( newval)
cmd YNetwork target 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.

<b>js</b>	function <b>set_wwwWatchdogDelay( newval)</b>
<b>node.js</b>	function <b>set_wwwWatchdogDelay( newval)</b>
<b>php</b>	function <b>set_wwwWatchdogDelay( \$newval)</b>
<b>cpp</b>	int <b>set_wwwWatchdogDelay( int newval)</b>
<b>m</b>	-(int) setWwwWatchdogDelay : (int) <b>newval</b>
<b>pas</b>	function <b>set_wwwWatchdogDelay( newval: LongInt): integer</b>
<b>vb</b>	function <b>set_wwwWatchdogDelay( ByVal newval As Integer) As Integer</b>
<b>cs</b>	int <b>set_wwwWatchdogDelay( int newval)</b>
<b>java</b>	int <b>set_wwwWatchdogDelay( int newval)</b>
<b>py</b>	def <b>set_wwwWatchdogDelay( newval)</b>
<b>cmd</b>	<b>YNetwork target set_wwwWatchdogDelay newval</b>

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.

```

js function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
nodejs function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
php function useDHCP( $fallbackIpAddr, $fallbackSubnetMaskLen, $fallbackRouter)
cpp int useDHCP( string fallbackIpAddr,
                 int fallbackSubnetMaskLen,
                 string fallbackRouter)
m -(int) useDHCP : (NSString*) fallbackIpAddr
                  : (int) fallbackSubnetMaskLen
                  : (NSString*) fallbackRouter
pas function useDHCP( fallbackIpAddr: string,
                      fallbackSubnetMaskLen: LongInt,
                      fallbackRouter: string): integer
vb function useDHCP( ByVal fallbackIpAddr As String,
                     ByVal fallbackSubnetMaskLen As Integer,
                     ByVal fallbackRouter As String) As Integer
cs int useDHCP( string fallbackIpAddr,
                 int fallbackSubnetMaskLen,
                 string fallbackRouter)
java int useDHCP( String fallbackIpAddr,
                  int fallbackSubnetMaskLen,
                  String fallbackRouter)
py def useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
cmd YNetwork target useDHCP fallbackIpAddr fallbackSubnetMaskLen fallbackRouter

```

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).

```

js function useStaticIP( ipAddress, subnetMaskLen, router)
nodejs function useStaticIP( ipAddress, subnetMaskLen, router)
php function useStaticIP( $ipAddress, $subnetMaskLen, $router)
cpp int useStaticIP( string ipAddress,
                     int subnetMaskLen,
                     string router)

m -(int) useStaticIP : (NSString*) ipAddress
                      : (int) subnetMaskLen
                      : (NSString*) router

pas function useStaticIP( ipAddress: string,
                          subnetMaskLen: LongInt,
                          router: string): integer

vb function useStaticIP( ByVal ipAddress As String,
                        ByVal subnetMaskLen As Integer,
                        ByVal router As String) As Integer

cs int useStaticIP( string ipAddress,
                    int subnetMaskLen,
                    string router)

java int useStaticIP( String ipAddress,
                      int subnetMaskLen,
                      String router)

py def useStaticIP( ipAddress, subnetMaskLen, router)
cmd YNetwork target useStaticIP ipAddress subnetMaskLen router

```

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()

YNetwork

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()****YOsControl****yFindOsControl() YOsControl.FindOsControl()**

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

js	function <b>yFindOsControl( func)</b>
node.js	function <b>FindOsControl( func)</b>
php	function <b>yFindOsControl( \$func)</b>
cpp	YOsControl* <b>yFindOsControl( const string&amp; func)</b>
m	YOsControl* <b>yFindOsControl( NSString* func)</b>
pas	function <b>yFindOsControl( func: string): TYOsControl</b>
vb	function <b>yFindOsControl( ByVal func As String) As YOsControl</b>
cs	YOsControl <b>FindOsControl( string func)</b>
java	YOsControl <b>FindOsControl( String func)</b>
py	def <b>FindOsControl( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'OS soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YOsControl.isOnline()` pour tester si le contrôle d'OS est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

**Retourne :**

un objet de classe `YOsControl` qui permet ensuite de contrôler le contrôle d'OS.

**YOsControl.FirstOsControl()****YOsControl****yFirstOsControl() YOsControl.FirstOsControl()**

Commence l'énumération des contrôle d'OS accessibles par la librairie.

js	function <b>yFirstOsControl( )</b>
node.js	function <b>FirstOsControl( )</b>
php	function <b>yFirstOsControl( )</b>
cpp	YOsControl* <b>yFirstOsControl( )</b>
m	YOsControl* <b>yFirstOsControl( )</b>
pas	function <b>yFirstOsControl( ): TYOsControl</b>
vb	function <b>yFirstOsControl( ) As YOsControl</b>
cs	YOsControl <b>FirstOsControl( )</b>
java	YOsControl <b>FirstOsControl( )</b>
py	def <b>FirstOsControl( )</b>

Utiliser la fonction `YOsControl.nextOsControl( )` pour itérer sur les autres contrôle d'OS.

**Retourne :**

un pointeur sur un objet `YOsControl`, correspondant à le premier contrôle d'OS accessible en ligne, ou `null` si il n'y a pas de contrôle d'OS disponibles.

**oscontrol→describe()oscontrol.describe()****YOsControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format TYPE ( NAME )=SERIAL.FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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'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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YOsControl target get_advertisedValue
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**oscontrol→get\_errorMessage()**  
**oscontrol→errorMessage()**  
**oscontrol.get\_errorMessage()****YOsControl**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

js	function <b>get_errorMessage( )</b>
nodejs	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	String <b>get_errorMessage( )</b>
py	def <b>get_errorMessage( )</b>

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**oscontrol→get\_functionId()****YOsControl****oscontrol→functionId()oscontrol.get\_functionId()**

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

```
js function get_hardwareId( )  
node.js function get_hardwareId( )  
php function get_hardwareId( )  
cpp string get_hardwareId( )  
m -(NSString*) hardwareId  
vb function get_hardwareId( ) As String  
cs string get_hardwareId( )  
java String get_hardwareId( )  
py def get_hardwareId( )
```

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.

js	function get_logicalName( )
node.js	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YOsControl target 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()**  
**oscontrol→module()oscontrol.get\_module()****YOsControl**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YOsControl**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

```
js function get_shutdownCountdown( )
nodejs function get_shutdownCountdown( )
php function get_shutdownCountdown( )
cpp int get_shutdownCountdown( )
m -(int) shutdownCountdown
pas function get_shutdownCountdown( ): LongInt
vb function get_shutdownCountdown( ) As Integer
cs int get_shutdownCountdown( )
java int get_shutdownCountdown( )
py def get_shutdownCountdown( )
cmd YOsControl target get_shutdownCountdown
```

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()

## YOsControl

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

js	function <b>load( msValidity)</b>
nodejs	function <b>load( msValidity)</b>
php	function <b>load( \$msValidity)</b>
cpp	YRETCODE <b>load( int msValidity)</b>
m	- <b>(YRETCODE) load : (int) msValidity</b>
pas	function <b>load( msValidity: integer): YRETCODE</b>
vb	function <b>load( ByVal msValidity As Integer) As YRETCODE</b>
cs	YRETCODE <b>load( int msValidity)</b>
java	int <b>load( long msValidity)</b>
py	def <b>load( msValidity)</b>

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()

## YOsControl

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextOsControl()</b>
node.js	function <b>nextOsControl()</b>
php	function <b>nextOsControl()</b>
cpp	YOsControl * <b>nextOsControl()</b>
m	-(YOsControl*) <b>nextOsControl</b>
pas	function <b>nextOsControl()</b> : TYOsControl
vb	function <b>nextOsControl()</b> As YOsControl
cs	YOsControl <b>nextOsControl()</b>
java	YOsControl <b>nextOsControl()</b>
py	def <b>nextOsControl()</b>

**Retourne :**

un pointeur sur un objet `YOsControl` accessible en ligne, ou `null` lorsque l'énumération est terminée.

## oscontrol→registerValueCallback() oscontrol.registerValueCallback()

YOsControl

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	<code>function registerValueCallback( callback)</code>
node.js	<code>function registerValueCallback( callback)</code>
php	<code>function registerValueCallback( \$callback)</code>
cpp	<code>int registerValueCallback( YOsControlValueCallback callback)</code>
m	<code>-(int) registerValueCallback : (YOsControlValueCallback) callback</code>
pas	<code>function registerValueCallback( callback: TYOsControlValueCallback): LongInt</code>
vb	<code>function registerValueCallback( ) As Integer</code>
cs	<code>int registerValueCallback( ValueCallback callback)</code>
java	<code>int registerValueCallback( UpdateCallback callback)</code>
py	<code>def registerValueCallback( callback)</code>

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**oscontrol→set\_logicalName()  
oscontrol→setLogicalName()  
oscontrol.set\_logicalName()****YOsControl**

Modifie le nom logique du contrôle d'OS.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YOsControl 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 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).

js	function set(userData( data)
node.js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

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.

```
js function shutdown( secBeforeShutDown)
nodejs function shutdown( secBeforeShutDown)
php function shutdown( $secBeforeShutDown)
cpp int shutdown( int secBeforeShutDown)
m -(int) shutdown : (int) secBeforeShutDown
pas function shutdown( secBeforeShutDown: LongInt): LongInt
vb function shutdown( ) As Integer
cs int shutdown( int secBeforeShutDown)
java int shutdown( int secBeforeShutDown)
py def shutdown( secBeforeShutDown)
cmd YOsControl target 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.

```
js function wait_async( callback, context)
nodejs 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(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

#### **power→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPower.FindPower()****YPower****yFindPower()YPower.FindPower()**

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

js	function <b>yFindPower( func)</b>
node.js	function <b>FindPower( func)</b>
php	function <b>yFindPower( \$func)</b>
cpp	YPower* <b>yFindPower( const string&amp; func)</b>
m	YPower* <b>yFindPower( NSString* func)</b>
pas	function <b>yFindPower( func: string): TYPower</b>
vb	function <b>yFindPower( ByVal func As String) As YPower</b>
cs	YPower <b>FindPower( string func)</b>
java	YPower <b>FindPower( String func)</b>
py	def <b>FindPower( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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()YPower.FirstPower()

YPower

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

```
js function yFirstPower( )
node.js function FirstPower()
php function yFirstPower( )
cpp YPower* yFirstPower( )
m YPower* yFirstPower( )
pas function yFirstPower( ): TYPower
vb function yFirstPower( ) As YPower
cs YPower FirstPower()
java YPower FirstPower()
py def FirstPower( )
```

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance électrique.

### Retourne :

un pointeur sur un objet `YPower`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)

m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YPower target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YPower <b>target get_advertisedValue</b>

**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).

```
js function get_cosPhi( )
nodejs function get_cosPhi( )
php function get_cosPhi( )
cpp double get_cosPhi( )
m -(double) cosPhi
pas function get_cosPhi( ): double
vb function get_cosPhi( ) As Double
cs double get_cosPhi( )
java double get_cosPhi( )
py def get_cosPhi( )
cmd YPower target 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 rentrée par le capteur (sans arrondi ni calibration).

js	function <b>get_currentRawValue( )</b>
node.js	function <b>get_currentRawValue( )</b>
php	function <b>get_currentRawValue( )</b>
cpp	double <b>get_currentRawValue( )</b>
m	-(double) currentRawValue
pas	function <b>get_currentRawValue( ): double</b>
vb	function <b>get_currentRawValue( ) As Double</b>
cs	double <b>get_currentRawValue( )</b>
java	double <b>get_currentRawValue( )</b>
py	def <b>get_currentRawValue( )</b>
cmd	YPower <b>target get_currentRawValue</b>

**Retourne :**

une valeur numérique représentant la valeur brute rentrée 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.

**js** function **get\_currentValue( )****node.js** function **get\_currentValue( )****php** function **get\_currentValue( )****cpp** double **get\_currentValue( )****m** -(double) currentValue**pas** function **get\_currentValue( ): double****vb** function **get\_currentValue( ) As Double****cs** double **get\_currentValue( )****java** double **get\_currentValue( )****py** def **get\_currentValue( )****cmd** YPower target **get\_currentValue****Retourne :**

une valeur numérique représentant la valeur instantanée de la puissance électrique

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**power→get\_errorMessage()****YPower****power→errorMessage()power.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
js function get_errorMessage( )  
nodejs function get_errorMessage( )  
php function get_errorMessage( )  
cpp string get_errorMessage( )  
m -(NSString*) errorMessage  
pas function get_errorMessage( ): string  
vb function get_errorMessage( ) As String  
cs string get_errorMessage( )  
java String get_errorMessage( )  
py def get_errorMessage( )
```

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()  
power→errorType()power.get\_errorType()****YPower**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function <b>get_friendlyName( )</b>
nodejs	function <b>get_friendlyName( )</b>
php	function <b>get_friendlyName( )</b>
cpp	string <b>get_friendlyName( )</b>
m	- <b>(NSString*) friendlyName</b>
cs	string <b>get_friendlyName( )</b>
java	<b>String get_friendlyName( )</b>
py	<b>def get_friendlyName( )</b>

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.

js	function <b>get_functionDescriptor()</b>
nodejs	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) <b>functionDescriptor</b>
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

js	function <b>get_hardwareId( )</b>
nodejs	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de puissance électrique (par exemple RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

## power→get\_highestValue() power→highestValue()power.get\_highestValue()

YPower

Retourne la valeur maximale observée pour la puissance électrique.

```
js function get_highestValue( )
nodejs function get_highestValue( )
php function get_highestValue( )
cpp double get_highestValue( )
m -(double) highestValue
pas function get_highestValue( ): double
vb function get_highestValue( ) As Double
cs double get_highestValue( )
java double get_highestValue( )
py def get_highestValue( )
cmd YPower target get_highestValue
```

**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.

```
js function get_logFrequency( )  
nodejs function get_logFrequency( )  
php function get_logFrequency( )  
cpp string get_logFrequency( )  
m -(NSString*) logFrequency  
pas function get_logFrequency( ):string  
vb function get_logFrequency( ) As String  
cs string get_logFrequency( )  
java String get_logFrequency( )  
py def get_logFrequency( )  
cmd YPower target get_logFrequency
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**power→get\_logicalName()****YPower****power→logicalName()power.get\_logicalName()**

Retourne le nom logique du capteur de puissance électrique.

```
js function get_logicalName( )
nodejs function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YPower target 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.

**js** function **get\_lowestValue( )****node.js** function **get\_lowestValue( )****php** function **get\_lowestValue( )****cpp** double **get\_lowestValue( )****m** -(double) lowestValue**pas** function **get\_lowestValue( )**: double**vb** function **get\_lowestValue( )** As Double**cs** double **get\_lowestValue( )****java** double **get\_lowestValue( )****py** def **get\_lowestValue( )****cmd** YPower target **get\_lowestValue****Retourne :**

une valeur numérique représentant la valeur minimale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne **Y\_LOWESTVALUE\_INVALID**.

**power→get\_meter()**

YPower

**power→meter()power.get\_meter()**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

```
js function get_meter( )
nodejs function get_meter( )
php function get_meter( )
cpp double get_meter( )
m -(double) meter
pas function get_meter( ): double
vb function get_meter( ) As Double
cs double get_meter( )
java double get_meter( )
py def get_meter( )
cmd YPower target 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

```
js function get_meterTimer( )  
node.js function get_meterTimer( )  
php function get_meterTimer( )  
cpp int get_meterTimer( )  
m -(int) meterTimer  
pas function get_meterTimer( ): LongInt  
vb function get_meterTimer( ) As Integer  
cs int get_meterTimer( )  
java int get_meterTimer( )  
py def get_meterTimer( )  
cmd YPower target 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.

js	function <b>get_module()</b>
nodejs	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**une instance de `YModule`

## power→get\_module\_async() power→module\_async()

YPower

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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.

## power→get\_recordedData() power→recordedData()power.get\_recordedData()

YPower

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YPower target 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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YPower target get_reportFrequency
```

**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.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YPower target 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.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YPower target get_unit</b>

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()

YPower

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)
cmd  YPower target loadCalibrationPoints rawValues refValues

```

### 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()

YPower

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextPower()</b>
nodejs	function <b>nextPower()</b>
php	function <b>nextPower()</b>
cpp	YPower * <b>nextPower()</b>
m	-(YPower*) <b>nextPower</b>
pas	function <b>nextPower()</b> : TYPower
vb	function <b>nextPower()</b> As YPower
cs	YPower <b>nextPower()</b>
java	YPower <b>nextPower()</b>
py	def <b>nextPower()</b>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YPowerTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YPowerTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYPowerTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

## power→registerValueCallback() power.registerValueCallback()

YPower

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YPowerValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YPowerValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYPowerValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**power→reset()power.reset()**

YPower

Réinitialise le compteur d'énergie.

```
js function reset( )  
nodejs function reset( )  
php function reset( )  
cpp int reset( )  
m -(int) reset  
pas function reset( ): LongInt  
vb function reset( ) As Integer  
cs int reset( )  
java int reset( )  
py def reset( )  
cmd YPower target 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.

js	function <b>set_highestValue( newval)</b>
nodejs	function <b>set_highestValue( newval)</b>
php	function <b>set_highestValue( \$newval)</b>
cpp	int <b>set_highestValue( double newval)</b>
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue( newval: double): integer</b>
vb	function <b>set_highestValue( ByVal newval As Double) As Integer</b>
cs	int <b>set_highestValue( double newval)</b>
java	int <b>set_highestValue( double newval)</b>
py	def <b>set_highestValue( newval)</b>
cmd	<b>YPower target set_highestValue newval</b>

**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.

```
js function set_logFrequency( newval)
node.js function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YPower target 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.

js	function <b>set_logicalName( newval)</b>
nodejs	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) setLogicalName : (NSString*) <b>newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	<b>YPower target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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.

```
js function set_lowestValue( newval)
node.js function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YPower target 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.

js	function <b>set_reportFrequency( newval)</b>
node.js	function <b>set_reportFrequency( newval)</b>
php	function <b>set_reportFrequency( \$newval)</b>
cpp	int <b>set_reportFrequency( const string&amp; newval)</b>
m	-(int) <b>setReportFrequency : (NSString*) newval</b>
pas	function <b>set_reportFrequency( newval: string): integer</b>
vb	function <b>set_reportFrequency( ByVal newval As String) As Integer</b>
cs	int <b>set_reportFrequency( string newval)</b>
java	int <b>set_reportFrequency( String newval)</b>
py	def <b>set_reportFrequency( newval)</b>
cmd	YPower <b>target set_reportFrequency newval</b>

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() power→setResolution()power.set\_resolution()

YPower

Modifie la résolution des valeurs mesurées.

```
js function set_resolution( newval)
node.js function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YPower target 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).

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	- <b>(void) set(userData : (void*) data)</b>
pas	procedure <b>set(userData Tobject data)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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()

YPower

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()****YPressure****yFindPressure()YPressure.FindPressure()**

Permet de retrouver un capteur de pression d'après un identifiant donné.

<b>js</b>	function <b>yFindPressure( func)</b>
<b>node.js</b>	function <b>FindPressure( func)</b>
<b>php</b>	function <b>yFindPressure( \$func)</b>
<b>cpp</b>	YPressure* <b>yFindPressure( const string&amp; func)</b>
<b>m</b>	YPressure* <b>yFindPressure( NSString* func)</b>
<b>pas</b>	function <b>yFindPressure( func: string): TYPressure</b>
<b>vb</b>	function <b>yFindPressure( ByVal func As String) As YPressure</b>
<b>cs</b>	YPressure <b>FindPressure( string func)</b>
<b>java</b>	YPressure <b>FindPressure( String func)</b>
<b>py</b>	def <b>FindPressure( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnLine()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de pression sans ambiguïté

**Retourne :**

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

## YPressure.FirstPressure()

### yFirstPressure() YPressure.FirstPressure()

## YPressure

Commence l'énumération des capteurs de pression accessibles par la librairie.

js	function <b>yFirstPressure()</b>
nodejs	function <b>FirstPressure()</b>
php	function <b>yFirstPressure()</b>
cpp	YPressure* <b>yFirstPressure()</b>
m	YPressure* <b>yFirstPressure()</b>
pas	function <b>yFirstPressure()</b> : TYPressure
vb	function <b>yFirstPressure()</b> As YPressure
cs	YPressure <b>FirstPressure()</b>
java	YPressure <b>FirstPressure()</b>
py	def <b>FirstPressure()</b>

Utiliser la fonction `YPressure.nextPressure()` pour itérer sur les autres capteurs de pression.

#### Retourne :

un pointeur sur un objet `YPressure`, correspondant à le premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

## pressure→calibrateFromPoints() pressure.calibrateFromPoints()

YPressure

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php   function calibrateFromPoints( $rawValues, $refValues)
cpp    int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)
m     -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                           refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YPressure target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YPressure target 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 rentrée par le capteur (sans arrondi ni calibration).

js	function <b>get_currentRawValue( )</b>
node.js	function <b>get_currentRawValue( )</b>
php	function <b>get_currentRawValue( )</b>
cpp	double <b>get_currentRawValue( )</b>
m	-(double) currentRawValue
pas	function <b>get_currentRawValue( ): double</b>
vb	function <b>get_currentRawValue( ) As Double</b>
cs	double <b>get_currentRawValue( )</b>
java	double <b>get_currentRawValue( )</b>
py	def <b>get_currentRawValue( )</b>
cmd	YPressure <b>target get_currentRawValue</b>

**Retourne :**

une valeur numérique représentant la valeur brute rentrée 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.

```
js function get_currentValue( )
node.js function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YPressure target get_currentValue
```

**Retourne :**

une valeur numérique représentant la mesure actuelle de la pression

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**pressure→getErrorMessage()  
pressure→errorMessage()  
pressure.getErrorMessage()****YPressure**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

js	function <b>getErrorMessage( )</b>
nodejs	function <b>getErrorMessage( )</b>
php	function <b>getErrorMessage( )</b>
cpp	string <b>getErrorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>getErrorMessage( )</b> : string
vb	function <b>getErrorMessage( )</b> As String
cs	string <b>getErrorMessage( )</b>
java	String <b>getErrorMessage( )</b>
py	def <b>getErrorMessage( )</b>

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

---

<b>pressure→get_functionDescriptor()</b>	<b>YPressure</b>
<b>pressure→functionDescriptor()</b>	
<b>pressure.get_functionDescriptor()</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

<code>js</code>	function <b>get_functionDescriptor()</b>
<code>nodejs</code>	function <b>get_functionDescriptor()</b>
<code>php</code>	function <b>get_functionDescriptor()</b>
<code>cpp</code>	YFUN_DESCR <b>get_functionDescriptor()</b>
<code>m</code>	-(YFUN_DESCR) <b>functionDescriptor</b>
<code>pas</code>	function <b>get_functionDescriptor()</b> : YFUN_DESCR
<code>vb</code>	function <b>get_functionDescriptor()</b> As YFUN_DESCR
<code>cs</code>	YFUN_DESCR <b>get_functionDescriptor()</b>
<code>java</code>	String <b>get_functionDescriptor()</b>
<code>py</code>	<b>def get_functionDescriptor()</b>

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.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

js	function get.hardwareId( )
node.js	function get.hardwareId( )
php	function get.hardwareId( )
cpp	string get.hardwareId( )
m	-(NSString*) hardwareId
vb	function get.hardwareId( ) As String
cs	string get.hardwareId( )
java	String get.hardwareId( )
py	def get.hardwareId( )

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.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	<b>def get_highestValue( )</b>
cmd	YPressure <b>target get_highestValue</b>

**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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YPressure target get_logFrequency
```

**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.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YPressure target 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.

**js** function **get\_lowestValue( )****node.js** function **get\_lowestValue( )****php** function **get\_lowestValue( )****cpp** double **get\_lowestValue( )****m** -(double) lowestValue**pas** function **get\_lowestValue( ): double****vb** function **get\_lowestValue( ) As Double****cs** double **get\_lowestValue( )****java** double **get\_lowestValue( )****py** def **get\_lowestValue( )****cmd** YPressure target **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.

js	<code>function get_module( )</code>
nodejs	<code>function get_module( )</code>
php	<code>function get_module( )</code>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	<code>function get_module( ): TYModule</code>
vb	<code>function get_module( ) As YModule</code>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

## pressure→get\_module\_async() pressure→module\_async()

YPressure

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m     -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs    YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YPressure target 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>pressure→get_reportFrequency()</b>	<b>YPressure</b>
<b>pressure→reportFrequency()</b>	
<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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YPressure target get_reportFrequency
```

**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.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YPressure target get_resolution
```

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.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YPressure target get_unit</b>

**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).

js	function get(userData)
nodejs	function get(userData)
php	function get(userData)
cpp	void * get(userData)
m	-(void*) userData
pas	function get(userData): Tobject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

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.

js	function <b>isOnline( )</b>
nodejs	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	def <b>isOnline( )</b>

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()

## YPressure

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## pressure→loadCalibrationPoints() pressure.loadCalibrationPoints()

YPressure

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)

cmd  YPressure target 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()

YPressure

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextPressure()</b>
nodejs	function <b>nextPressure()</b>
php	function <b>nextPressure()</b>
cpp	YPressure * <b>nextPressure()</b>
m	-(YPressure*) <b>nextPressure</b>
pas	function <b>nextPressure()</b> : TYPressure
vb	function <b>nextPressure()</b> As YPressure
cs	YPressure <b>nextPressure()</b>
java	YPressure <b>nextPressure()</b>
py	def <b>nextPressure()</b>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YPressureTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YPressureTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYPRESSURETIMEDREPORTCALLBACK): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

## pressure→registerValueCallback() pressure.registerValueCallback()

YPressure

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YPressureValueCallback callback)
m -(int) registerValueCallback : (YPressureValueCallback) callback
pas function registerValueCallback( callback: TYPressureValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pressure→set\_highestValue()**  
**pressure→setHighestValue()**  
**pressure.set\_highestValue()**

**YPressure**

Modifie la mémoire de valeur maximale observée pour la pression.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YPressure target 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.

js	function <b>set_logFrequency( newval)</b>
node.js	function <b>set_logFrequency( newval)</b>
php	function <b>set_logFrequency( \$newval)</b>
cpp	int <b>set_logFrequency( const string&amp; newval)</b>
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency( newval: string): integer</b>
vb	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
cs	int <b>set_logFrequency( string newval)</b>
java	int <b>set_logFrequency( String newval)</b>
py	def <b>set_logFrequency( newval)</b>
cmd	YPressure <b>target set_logFrequency newval</b>

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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YPressure 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 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.

js	function <b>set_lowestValue( newval)</b>
node.js	function <b>set_lowestValue( newval)</b>
php	function <b>set_lowestValue( \$newval)</b>
cpp	int <b>set_lowestValue( double newval)</b>
m	-(int) <b>setLowestValue : (double) newval</b>
pas	function <b>set_lowestValue( newval: double): integer</b>
vb	function <b>set_lowestValue( ByVal newval As Double) As Integer</b>
cs	int <b>set_lowestValue( double newval)</b>
java	int <b>set_lowestValue( double newval)</b>
py	def <b>set_lowestValue( newval)</b>
cmd	YPressure <b>target set_lowestValue newval</b>

#### 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.

```
js function set_reportFrequency( newval)
nodejs function set_reportFrequency( newval)
php function set_reportFrequency( $newval)
cpp int set_reportFrequency( const string& newval)
m -(int) setReportFrequency : (NSString*) newval
pas function set_reportFrequency( newval: string): integer
vb function set_reportFrequency( ByVal newval As String) As Integer
cs int set_reportFrequency( string newval)
java int set_reportFrequency( String newval)
py def set_reportFrequency( newval)
cmd YPressure target 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.

<code>js</code>	<code>function set_resolution( newval)</code>
<code>nodejs</code>	<code>function set_resolution( newval)</code>
<code>php</code>	<code>function set_resolution( \$newval)</code>
<code>cpp</code>	<code>int set_resolution( double newval)</code>
<code>m</code>	<code>-(int) setResolution : (double) newval</code>
<code>pas</code>	<code>function set_resolution( newval: double): integer</code>
<code>vb</code>	<code>function set_resolution( ByVal newval As Double) As Integer</code>
<code>cs</code>	<code>int set_resolution( double newval)</code>
<code>java</code>	<code>int set_resolution( double newval)</code>
<code>py</code>	<code>def set_resolution( newval)</code>
<code>cmd</code>	<code>YPressure target set_resolution newval</code>

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).

```
js function set(userData( data)
nodejs function set(userData( data)
php function set(userData( $data)
cpp void set(userData( void* data)
m -(void) setUserData : (void*) data
pas procedure set(userData( data: Tobject)
vb procedure set(userData( ByVal data As Object)
cs void set(userData( object data)
java void set(userData( Object data)
py def set(userData( data)
```

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()

## YPressure

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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;
require_once('yocto_pwmoutput.php');
#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 \$FUNCTION\$ sous la forme d'un nombre à virgule entre 0 et 1.

#### **pwmoutput→get\_dutyCycleAtPowerOn()**

Retourne le duty cycle du \$FUNCTION\$ au démarrage du module, sous la forme d'un nombre à virgule entre 0.0% et 100.

#### **pwmoutput→get\_enabled()**

Retourne l'état de fonctionnement du \$FUNCTION\$.

#### **pwmoutput→get\_enabledAtPowerOn()**

Retourne l'état de fonctionnement du \$FUNCTION\$ 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 \$FUNCTION\$ en Hz.

#### **pwmoutput→get\_friendlyName()**

Retourne un identifiant global du PWM au format NOM\_MODULE . NOM\_FONCTION.

**pwmoutput→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**pwmoutput→get\_functionId()**

Retourne l'identifiant matériel du PWM, sans référence au module.

**pwmoutput→get\_hardwareId()**

Retourne l'identifiant matériel unique du PWM au format SERIAL . FUNCTIONID.

**pwmoutput→get\_logicalName()**

Retourne le nom logique du PWM.

**pwmoutput→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**pwmoutput→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**pwmoutput→get\_period()**

Retourne la période du \$FUNCTION\$ en nano secondes.

**pwmoutput→get\_pulseDuration()**

Retourne la longueur d'une impulsion du \$FUNCTION\$ en millisecondes.

**pwmoutput→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**pwmoutput→isOnline()**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

**pwmoutput→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

**pwmoutput→load(msValidity)**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

**pwmoutput→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

**pwmoutput→nextPwmOutput()**

Continue l'énumération des PWM commencée à l'aide de yFirstPwmOutput( ).

**pwmoutput→pulseDurationMove(ms\_target, ms\_duration)**

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

**pwmoutput→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**pwmoutput→set\_dutyCycle(newval)**

Configure le duty cycle du \$FUNCTION\$.

**pwmoutput→set\_dutyCycleAtPowerOn(newval)**

Configure le duty cycle du \$FUNCTION\$ au démarrage du module.

**pwmoutput→set\_enabled(newval)**

Démarre ou arrête le \$FUNCTION\$.

**pwmoutput→set\_enabledAtPowerOn(newval)**

Configure l'état du fonctionnement du \$FUNCTION\$ à la mise sous tension du module.

**pwmoutput→set\_frequency(newval)**

Configure la fréquence du \$FUNCTION\$.

**pwmoutput→set\_logicalName(newval)**

Modifie le nom logique du PWM.

**pwmoutput→set\_period(newval)**

### 3. Reference

---

Configure la période du \$FUNCTION\$.

**pwmoutput→set\_pulseDuration(newval)**

Configure la longueur d'une impulsion du \$FUNCTION\$.

**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()YPwmOutput.FindPwmOutput()****YPwmOutput**

Permet de retrouver un PWM d'après un identifiant donné.

<code>js</code>	<code>function yFindPwmOutput( func)</code>
<code>nodejs</code>	<code>function FindPwmOutput( func)</code>
<code>php</code>	<code>function yFindPwmOutput( \$func)</code>
<code>cpp</code>	<code>YPwmOutput* yFindPwmOutput( const string&amp; func)</code>
<code>m</code>	<code>YPwmOutput* yFindPwmOutput( NSString* func)</code>
<code>pas</code>	<code>function yFindPwmOutput( func: string): TYPwmOutput</code>
<code>vb</code>	<code>function yFindPwmOutput( ByVal func As String) As YPwmOutput</code>
<code>cs</code>	<code>YPwmOutput FindPwmOutput( string func)</code>
<code>java</code>	<code>YPwmOutput FindPwmOutput( String func)</code>
<code>py</code>	<code>def FindPwmOutput( 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 le PWM soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmOutput.isOnline()` pour tester si le PWM est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

`func` une chaîne de caractères qui référence le PWM sans ambiguïté

**Retourne :**

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

**YPwmOutput.FirstPwmOutput()****YPwmOutput****yFirstPwmOutput()YPwmOutput.FirstPwmOutput()**

Commence l'énumération des PWM accessibles par la librairie.

js	function <b>yFirstPwmOutput( )</b>
node.js	function <b>FirstPwmOutput( )</b>
php	function <b>yFirstPwmOutput( )</b>
cpp	YPwmOutput* <b>yFirstPwmOutput( )</b>
m	YPwmOutput* <b>yFirstPwmOutput( )</b>
pas	function <b>yFirstPwmOutput( ): TYPwmOutput</b>
vb	function <b>yFirstPwmOutput( ) As YPwmOutput</b>
cs	YPwmOutput <b>FirstPwmOutput( )</b>
java	YPwmOutput <b>FirstPwmOutput( )</b>
py	def <b>FirstPwmOutput( )</b>

Utiliser la fonction `YPwmOutput .nextPwmOutput( )` pour itérer sur les autres PWM.

**Retourne :**

un pointeur sur un objet `YPwmOutput`, correspondant à le 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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.

```
js function dutyCycleMove( target, ms_duration)
node.js function dutyCycleMove( target, ms_duration)
php function dutyCycleMove( $target, $ms_duration)
cpp int dutyCycleMove( double target, int ms_duration)
m -(int) dutyCycleMove : (double) target : (int) ms_duration
pas function dutyCycleMove( target: double, ms_duration: LongInt): LongInt
vb function dutyCycleMove( ) As Integer
cs int dutyCycleMove( double target, int ms_duration)
java int dutyCycleMove( double target, int ms_duration)
py def dutyCycleMove( target, ms_duration)
cmd YPwmOutput target 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) advertisedValue
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YPwmOutput <b>target get_advertisedValue</b>

**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 \$FUNCTION\$ sous la forme d'un nombre à virgule entre 0 et 1.

js	function <b>get_dutyCycle( )</b>
node.js	function <b>get_dutyCycle( )</b>
php	function <b>get_dutyCycle( )</b>
cpp	double <b>get_dutyCycle( )</b>
m	-{double} dutyCycle
pas	function <b>get_dutyCycle( )</b> : double
vb	function <b>get_dutyCycle( )</b> As Double
cs	double <b>get_dutyCycle( )</b>
java	double <b>get_dutyCycle( )</b>
py	def <b>get_dutyCycle( )</b>
cmd	<b>YPwmOutput target get_dutyCycle</b>

**Retourne :**

une valeur numérique représentant le duty cycle du \$FUNCTION\$ sous la forme d'un nombre à virgule entre 0 et 1

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 \$FUNCTION\$ au démarrage du module, sous la forme d'un nombre à virgule entre 0.0% et 100.

```
js function get_dutyCycleAtPowerOn( )
nodejs function get_dutyCycleAtPowerOn( )
php function get_dutyCycleAtPowerOn( )
cpp double get_dutyCycleAtPowerOn( )
m -(double) dutyCycleAtPowerOn
pas function get_dutyCycleAtPowerOn( ): double
vb function get_dutyCycleAtPowerOn( ) As Double
cs double get_dutyCycleAtPowerOn( )
java double get_dutyCycleAtPowerOn( )
py def get_dutyCycleAtPowerOn( )
cmd YPwmOutput target get_dutyCycleAtPowerOn
```

0%

**Retourne :**

une valeur numérique représentant le duty cycle du \$FUNCTION\$ au démarrage du module, sous la forme d'un nombre à virgule entre 0.0% et 100

En cas d'erreur, déclenche une exception ou retourne Y\_DUTYCYCLEATPOWERON\_INVALID.

**pwmoutput→get\_enabled()** YPwmOutput  
**pwmoutput→enabled()pwmoutput.get\_enabled()**

---

Retourne l'état de fonctionnement du \$FUNCTION\$.

```
js function get_enabled( )
node.js function get_enabled( )
php function get_enabled( )
cpp Y_ENABLED_enum get_enabled( )
m -(Y_ENABLED_enum) enabled
pas function get_enabled( ): Integer
vb function get_enabled( ) As Integer
cs int get_enabled( )
java int get_enabled( )
py def get_enabled( )
cmd YPwmOutput target 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.

**pwmoutput→get\_enabledAtPowerOn()**  
**pwmoutput→enabledAtPowerOn()**  
**pwmoutput.get\_enabledAtPowerOn()**

**YPwmOutput**

Retourne l'état de fonctionnement du \$FUNCTION\$ à la mise sous tension du module.

js	function <b>get_enabledAtPowerOn( )</b>
nodejs	function <b>get_enabledAtPowerOn( )</b>
php	function <b>get_enabledAtPowerOn( )</b>
cpp	Y_ENABLEDATPOWERON_enum <b>get_enabledAtPowerOn( )</b>
m	-(Y_ENABLEDATPOWERON_enum) enabledAtPowerOn
pas	function <b>get_enabledAtPowerOn( ): Integer</b>
vb	function <b>get_enabledAtPowerOn( ) As Integer</b>
cs	int <b>get_enabledAtPowerOn( )</b>
java	int <b>get_enabledAtPowerOn( )</b>
py	def <b>get_enabledAtPowerOn( )</b>
cmd	YPwmOutput <b>target get_enabledAtPowerOn</b>

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état de fonctionnement du \$FUNCTION\$ 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.

js	function getErrorMessage( )
nodejs	function getErrorMessage( )
php	function getErrorMessage( )
cpp	string getErrorMessage( )
m	-(NSString*) errorMessage
pas	function getErrorMessage( ): string
vb	function getErrorMessage( ) As String
cs	string getErrorMessage( )
java	String getErrorMessage( )
py	def getErrorMessage( )

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.

js	function <b>get_errorType( )</b>
nodejs	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	YRETCODE <b>get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	YRETCODE <b>get_errorType( )</b>
java	int <b>get_errorType( )</b>
py	def <b>get_errorType( )</b>

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 \$FUNCTION\$ en Hz.

js	function <b>get_frequency()</b> {
node.js	function <b>get_frequency()</b> {
php	function <b>get_frequency()</b> {
cpp	int <b>get_frequency()</b> {
m	-(int) frequency
pas	function <b>get_frequency()</b> : LongInt
vb	function <b>get_frequency()</b> As Integer
cs	int <b>get_frequency()</b> {
java	int <b>get_frequency()</b> {
py	def <b>get_frequency()</b> {
cmd	YPwmOutput target <b>get_frequency</b>

**Retourne :**

un entier représentant la fréquence du \$FUNCTION\$ 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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId( )</b>
nodejs	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	<b>String get_functionId( )</b>
py	<b>def get_functionId( )</b>

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.

js	function get.hardwareId( )
node.js	function get.hardwareId( )
php	function get.hardwareId( )
cpp	string get.hardwareId( )
m	-(NSString*) hardwareId
vb	function get.hardwareId( ) As String
cs	string get.hardwareId( )
java	String get.hardwareId( )
py	def get.hardwareId( )

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.

js	function get_logicalName( )
node.js	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YPwmOutput target 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.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()

YPwmOutput

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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 \$FUNCTION\$ en nano secondes.

js	function <b>get_period()</b>
node.js	function <b>get_period()</b>
php	function <b>get_period()</b>
cpp	double <b>get_period()</b>
m	-(double) period
pas	function <b>get_period()</b> : double
vb	function <b>get_period()</b> As Double
cs	double <b>get_period()</b>
java	double <b>get_period()</b>
py	def <b>get_period()</b>
cmd	<b>YPwmOutput target get_period</b>

**Retourne :**

une valeur numérique représentant la période du \$FUNCTION\$ en nano secondes

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 \$FUNCTION\$ en millisecondes.

js	function <b>get_pulseDuration( )</b>
node.js	function <b>get_pulseDuration( )</b>
php	function <b>get_pulseDuration( )</b>
cpp	double <b>get_pulseDuration( )</b>
m	-(double) pulseDuration
pas	function <b>get_pulseDuration( )</b> : double
vb	function <b>get_pulseDuration( )</b> As Double
cs	double <b>get_pulseDuration( )</b>
java	double <b>get_pulseDuration( )</b>
py	def <b>get_pulseDuration( )</b>
cmd	YPwmOutput <b>target get_pulseDuration</b>

**Retourne :**

une valeur numérique représentant la longueur d'une impulsion du \$FUNCTION\$ 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).

js	function <b>get(userData)</b> {
nodejs	function <b>get(userData)</b> {
php	function <b>get(userData)</b> {
cpp	void * <b>get(userData)</b> {
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b> {
java	Object <b>get(userData)</b> {
py	def <b>get(userData)</b> {

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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	<b>def isOnline( )</b>

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()

YPwmOutput

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()

YPwmOutput

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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.

**pwmoutput→nextPwmOutput()**  
**pwmoutput.nextPwmOutput()****YPwmOutput**Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

js	<code>function nextPwmOutput( )</code>
nodejs	<code>function nextPwmOutput( )</code>
php	<code>function nextPwmOutput( )</code>
cpp	<code>YPwmOutput * nextPwmOutput( )</code>
m	<code>-(YPwmOutput*) nextPwmOutput</code>
pas	<code>function nextPwmOutput( ): TYPwmOutput</code>
vb	<code>function nextPwmOutput( ) As YPwmOutput</code>
cs	<code>YPwmOutput nextPwmOutput( )</code>
java	<code>YPwmOutput nextPwmOutput( )</code>
py	<code>def nextPwmOutput( )</code>

**Retourne :**un pointeur sur un objet `YPwmOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pwmoutput→pulseDurationMove()  
pwmoutput.pulseDurationMove()****YPwmOutput**

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

```
js function pulseDurationMove( ms_target, ms_duration)
node.js function pulseDurationMove( ms_target, ms_duration)
php function pulseDurationMove( $ms_target, $ms_duration)
cpp int pulseDurationMove( double ms_target, int ms_duration)
m -(int) pulseDurationMove : (double) ms_target : (int) ms_duration
pas function pulseDurationMove( ms_target: double,
                                ms_duration: LongInt): LongInt
vb function pulseDurationMove( ) As Integer
cs int pulseDurationMove( double ms_target, int ms_duration)
java int pulseDurationMove( double ms_target, int ms_duration)
py def pulseDurationMove( ms_target, ms_duration)
cmd YPwmOutput target pulseDurationMove ms_target ms_duration
```

**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.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YPwmOutputValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YPwmOutputValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYPwmOutputValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pwmoutput→set\_dutyCycle()**  
**pwmoutput→setDutyCycle()**  
**pwmoutput.set\_dutyCycle()**

YPwmOutput

Configure le duty cycle du \$FUNCTION\$.

```
js function set_dutyCycle( newval)
nodejs function set_dutyCycle( newval)
php function set_dutyCycle( $newval)
cpp int set_dutyCycle( double newval)
m -(int) setDutyCycle : (double) newval
pas function set_dutyCycle( newval: double): integer
vb function set_dutyCycle( ByVal newval As Double) As Integer
cs int set_dutyCycle( double newval)
java int set_dutyCycle( double newval)
py def set_dutyCycle( newval)
cmd YPwmOutput target set_dutyCycle newval
```

**Paramètres :**

**newval** une valeur numérique

**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**

Configure le duty cycle du \$FUNCTION\$ au démarrage du module.

<b>js</b>	function <b>set_dutyCycleAtPowerOn( newval)</b>
<b>node.js</b>	function <b>set_dutyCycleAtPowerOn( newval)</b>
<b>php</b>	function <b>set_dutyCycleAtPowerOn( \$newval)</b>
<b>cpp</b>	int <b>set_dutyCycleAtPowerOn( double newval)</b>
<b>m</b>	-(int) setDutyCycleAtPowerOn : (double) <b>newval</b>
<b>pas</b>	function <b>set_dutyCycleAtPowerOn( newval: double): integer</b>
<b>vb</b>	function <b>set_dutyCycleAtPowerOn( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_dutyCycleAtPowerOn( double newval)</b>
<b>java</b>	int <b>set_dutyCycleAtPowerOn( double newval)</b>
<b>py</b>	def <b>set_dutyCycleAtPowerOn( newval)</b>
<b>cmd</b>	YPwmOutput <b>target set_dutyCycleAtPowerOn newval</b>

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur numérique

**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 \$FUNCTION\$.

```
js function set_enabled( newval)
node.js function set_enabled( newval)
php function set_enabled( $newval)
cpp int set_enabled( Y_ENABLED_enum newval)
m -(int) setEnabled : (Y_ENABLED_enum) newval
pas function set_enabled( newval: Integer): integer
vb function set_enabled( ByVal newval As Integer) As Integer
cs int set_enabled( int newval)
java int set_enabled( int newval)
py def set_enabled( newval)
cmd YPwmOutput target 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**

Configure l'état du fonctionnement du \$FUNCTION\$ à la mise sous tension du module.

js	function <b>set_enabledAtPowerOn( newval)</b>
node.js	function <b>set_enabledAtPowerOn( newval)</b>
php	function <b>set_enabledAtPowerOn( \$newval)</b>
cpp	int <b>set_enabledAtPowerOn( Y_ENABLEDATPOWERON_enum newval)</b>
m	-(int) <b>setEnabledAtPowerOn : (Y_ENABLEDATPOWERON_enum) newval</b>
pas	function <b>set_enabledAtPowerOn( newval: Integer): integer</b>
vb	function <b>set_enabledAtPowerOn( ByVal newval As Integer) As Integer</b>
cs	int <b>set_enabledAtPowerOn( int newval)</b>
java	int <b>set_enabledAtPowerOn( int newval)</b>
py	def <b>set_enabledAtPowerOn( newval)</b>
cmd	YPwmOutput <b>target set_enabledAtPowerOn newval</b>

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.

**pwmoutput→set\_frequency()  
pwmoutput→setFrequency()  
pwmoutput.set\_frequency()****YPwmOutput**

Configure la fréquence du \$FUNCTION\$.

js	function set_frequency( newval)
nodejs	function set_frequency( newval)
php	function set_frequency( \$newval)
cpp	int set_frequency( int newval)
m	-(int) setFrequency : (int) newval
pas	function set_frequency( newval: LongInt): integer
vb	function set_frequency( ByVal newval As Integer) As Integer
cs	int set_frequency( int newval)
java	int set_frequency( int newval)
py	def set_frequency( newval)
cmd	YPwmOutput target set_frequency newval

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

**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.

**pwmoutput→set\_logicalName()**  
**pwmoutput→setLogicalName()**  
**pwmoutput.set\_logicalName()**

YPwmOutput

Modifie le nom logique du PWM.

js	function <b>set_logicalName( newval)</b>
node.js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	- (int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	YPwmOutput <b>target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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()**

Configure la période du \$FUNCTION\$.

js	function <b>set_period( newval)</b>
node.js	function <b>set_period( newval)</b>
php	function <b>set_period( \$newval)</b>
cpp	int <b>set_period( double newval)</b>
m	-(int) setPeriod : (double) <b>newval</b>
pas	function <b>set_period( newval: double): integer</b>
vb	function <b>set_period( ByVal newval As Double) As Integer</b>
cs	int <b>set_period( double newval)</b>
java	int <b>set_period( double newval)</b>
py	def <b>set_period( newval)</b>
cmd	<b>YPwmOutput target set_period newval</b>

**Paramètres :**

**newval** une valeur numérique

**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**

Configure la longueur d'une impulsion du \$FUNCTION\$.

<b>js</b>	function <b>set_pulseDuration( newval)</b>
<b>node.js</b>	function <b>set_pulseDuration( newval)</b>
<b>php</b>	function <b>set_pulseDuration( \$newval)</b>
<b>cpp</b>	int <b>set_pulseDuration( double newval)</b>
<b>m</b>	-(int) <b>setPulseDuration : (double) newval</b>
<b>pas</b>	function <b>set_pulseDuration( newval: double): integer</b>
<b>vb</b>	function <b>set_pulseDuration( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_pulseDuration( double newval)</b>
<b>java</b>	int <b>set_pulseDuration( double newval)</b>
<b>py</b>	<b>def set_pulseDuration( newval)</b>
<b>cmd</b>	<b>YPwmOutput target set_pulseDuration newval</b>

Attention la longueur d'un impulsion ne peut pas être plus grande que la période, dans le cas contraire, la longueur sera automatiquement tronqué à la période.

**Paramètres :**

**newval** une valeur numérique

**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).

```
js function set(userData) {  
nodejs function set(userData) {  
php function set(userData) {  
cpp void set(userData) {  
m -(void) setUserData : (void*) data  
pas procedure set(userData: Tobject)  
vb procedure set(userData( ByVal data As Object)  
cs void set(userData( object data  
java void set(userData( Object data  
py def set(userData( data)
```

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.

```
js function wait_async( callback, context)
nodejs 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()****YPwmPowerSource.FindPwmPowerSource()****YPwmPowerSource**

Permet de retrouver une source de tension d'après un identifiant donné.

```
js function yFindPwmPowerSource( func)
nodejs function FindPwmPowerSource( func)
php function yFindPwmPowerSource( $func)
cpp YPwmPowerSource* yFindPwmPowerSource( const string& func)
m YPwmPowerSource* yFindPwmPowerSource( NSString* func)
pas function yFindPwmPowerSource( func: string): TYPwmPowerSource
vb function yFindPwmPowerSource( ByVal func As String) As YPwmPowerSource
cs YPwmPowerSource FindPwmPowerSource( string func)
java YPwmPowerSource FindPwmPowerSource( String func)
py def FindPwmPowerSource( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmPowerSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

**Retourne :**

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

**YPwmPowerSource.FirstPwmPowerSource()****YPwmPowerSource****yFirstPwmPowerSource()****YPwmPowerSource.FirstPwmPowerSource()**

Commence l'énumération des Source de tension accessibles par la librairie.

js	function yFirstPwmPowerSource( )
node.js	function FirstPwmPowerSource( )
php	function yFirstPwmPowerSource( )
cpp	YPwmPowerSource* yFirstPwmPowerSource( )
m	YPwmPowerSource* yFirstPwmPowerSource( )
pas	function yFirstPwmPowerSource( ): TYPwmPowerSource
vb	function yFirstPwmPowerSource( ) As YPwmPowerSource
cs	YPwmPowerSource FirstPwmPowerSource( )
java	YPwmPowerSource FirstPwmPowerSource( )
py	def FirstPwmPowerSource( )

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	- <b>(NSString*</b> ) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant la source de tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pwmpowersource→get\_advertisedValue()**  
**pwmpowersource→advertisedValue()**  
**pwmpowersource.get\_advertisedValue()**

**YPwmPowerSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
node.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 YPwmPowerSource 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.

**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.

js	function get_errorMessage( )
nodejs	function get_errorMessage( )
php	function get_errorMessage( )
cpp	string get_errorMessage( )
m	-(NSString*) errorMessage
pas	function get_errorMessage( ): string
vb	function get_errorMessage( ) As String
cs	string get_errorMessage( )
java	String get_errorMessage( )
py	def get_errorMessage( )

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.

js	function <b>get_errorType( )</b>
nodejs	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	<b>YRETCODE get_errorType( )</b>
pas	function <b>get_errorType( ): YRETCODE</b>
vb	function <b>get_errorType( ) As YRETCODE</b>
cs	<b>YRETCODE get_errorType( )</b>
java	<b>int get_errorType( )</b>
py	<b>def get_errorType( )</b>

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la 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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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.

js	function <b>get_functionId( )</b>
nodejs	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( )</b> As String
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la source de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pwmpowersource→get\_hardwareId()**  
**pwmpowersource→hardwareId()**  
**pwmpowersource.get\_hardwareId()**

**YPwmPowerSource**

Retourne l'identifiant matériel unique de la source de tension au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	<b>String get_hardwareId( )</b>
py	<b>def get_hardwareId( )</b>

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.

js	function get_logicalName( )
nodejs	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YPwmPowerSource 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.

**pwmpowersource→get\_module()**  
**pwmpowersource→module()**  
**pwmpowersource.get\_module()**

**YPwmPowerSource**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module()</b>
nodejs	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	YModule * <b>get_module()</b>
m	-(YModule*) module
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	YModule <b>get_module()</b>
java	YModule <b>get_module()</b>
py	def <b>get_module()</b>

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**pwmpowersource→get\_module\_async()**  
**pwmpowersource→module\_async()****YPwmPowerSource**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`js` `function get_module_async( callback, context)`  
`node.js` `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.

js	function <b>get_powerMode( )</b>
node.js	function <b>get_powerMode( )</b>
php	function <b>get_powerMode( )</b>
cpp	Y_POWERMODE_enum <b>get_powerMode( )</b>
m	-(Y_POWERMODE_enum) powerMode
pas	function <b>get_powerMode( )</b> : Integer
vb	function <b>get_powerMode( )</b> As Integer
cs	int <b>get_powerMode( )</b>
java	int <b>get_powerMode( )</b>
py	def <b>get_powerMode( )</b>

**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).

```
js function get(userData) 
nodejs function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	def <b>isOnline( )</b>

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()

## YPwmPowerSource

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()

## YPwmPowerSource

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

<code>js</code>	<code>function nextPwmPowerSource( )</code>
<code>nodejs</code>	<code>function nextPwmPowerSource( )</code>
<code>php</code>	<code>function nextPwmPowerSource( )</code>
<code>cpp</code>	<code>YPwmPowerSource * nextPwmPowerSource( )</code>
<code>m</code>	<code>-(YPwmPowerSource*) nextPwmPowerSource</code>
<code>pas</code>	<code>function nextPwmPowerSource( ): TYPwmPowerSource</code>
<code>vb</code>	<code>function nextPwmPowerSource( ) As YPwmPowerSource</code>
<code>cs</code>	<code>YPwmPowerSource nextPwmPowerSource( )</code>
<code>java</code>	<code>YPwmPowerSource nextPwmPowerSource( )</code>
<code>py</code>	<code>def nextPwmPowerSource( )</code>

**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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YPwmPowerSourceValueCallback callback)
m -(int) registerValueCallback : (YPwmPowerSourceValueCallback) callback
pas function registerValueCallback( callback: TYPwmPowerSourceValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YPwmPowerSource target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la source de tension.

**Retourne :**

`YAPI_SUCCESS` si l'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.

js	function <b>set_powerMode( newval)</b>
nodejs	function <b>set_powerMode( newval)</b>
php	function <b>set_powerMode( \$newval)</b>
cpp	int <b>set_powerMode( Y_POWERMODE_enum newval)</b>
m	- <b>(int) setPowerMode : (Y_POWERMODE_enum) newval</b>
pas	function <b>set_powerMode( newval: Integer): integer</b>
vb	function <b>set_powerMode( ByVal newval As Integer) As Integer</b>
cs	int <b>set_powerMode( int newval)</b>
java	int <b>set_powerMode( int newval)</b>
py	def <b>set_powerMode( newval)</b>
cmd	YPwmPowerSource <b>target set_powerMode newval</b>

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).

js	function set(userData( data)
node.js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

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()****YPwmPowerSource**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**js** `function wait_async( callback, context)`  
**node.js** `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()YQt.FindQt()

YQt

Permet de retrouver un élément de quaternion d'après un identifiant donné.

js	function <b>yFindQt( func)</b>
node.js	function <b>FindQt( func)</b>
php	function <b>yFindQt( \$func)</b>
cpp	<b>YQt* yFindQt( string func)</b>
m	<b>+ (YQt*) yFindQt : (NSString*) func</b>
pas	function <b>yFindQt( func: string): TYQt</b>
vb	function <b>yFindQt( ByVal func As String) As YQt</b>
cs	<b>YQt FindQt( string func)</b>
java	<b>YQt FindQt( String func)</b>
py	<b>def FindQt( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnline()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

### Retourne :

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

**YQt.FirstQt()****YQt****yFirstQt()YQt.FirstQt()**

Commence l'énumération des éléments de quaternion accessibles par la librairie.

js	function <b>yFirstQt( )</b>
nodejs	function <b>FirstQt( )</b>
php	function <b>yFirstQt( )</b>
cpp	YQt* <b>yFirstQt( )</b>
m	YQt* <b>yFirstQt( )</b>
pas	function <b>yFirstQt( ): TYQt</b>
vb	function <b>yFirstQt( ) As YQt</b>
cs	<b>YQt FirstQt( )</b>
java	<b>YQt FirstQt( )</b>
py	<b>def FirstQt( )</b>

Utiliser la fonction `YQt.nextQt( )` pour itérer sur les autres éléments de quaternion.

**Retourne :**

un pointeur sur un objet `YQt`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                           vector<double> refValues)

m   -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                           refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YSensor target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
node.js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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'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).

```
js function get_advertisedValue( )  
node.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 YSensor target 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() qt→currentRawValue()qt.get\_currentRawValue()

YQt

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
js function get_currentRawValue( )  
nodejs function get_currentRawValue( )  
php function get_currentRawValue( )  
cpp double get_currentRawValue( )  
m -(double) currentRawValue  
pas function get_currentRawValue( ): double  
vb function get_currentRawValue( ) As Double  
cs double get_currentRawValue( )  
java double get_currentRawValue( )  
py def get_currentRawValue( )  
cmd YSensor target get_currentRawValue
```

**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.

```
js function get_currentValue( )
node.js function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YSensor target 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() qt→errorMessage()qt.get\_errorMessage()

YQt

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
js function getErrorMessage( )  
nodejs function getErrorMessage( )  
php function getErrorMessage( )  
cpp string getErrorMessage( )  
m -(NSString*) errorMessage  
pas function getErrorMessage( ): string  
vb function getErrorMessage( ) As String  
cs string getErrorMessage( )  
java String getErrorMessage( )  
py def getErrorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

## qt→get\_errorType() qt→errorType()qt.get\_errorType()

YQt

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

Le chaîne renvoyée utilise soit les noms logiques du module et de l'élément de quaternion si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'élément de quaternion (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

**qt→get\_functionDescriptor()**  
**qt→functionDescriptor()qt.get\_functionDescriptor()**

YQt

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
js function get_functionDescriptor( )
node.js function get_functionDescriptor( )
php function get_functionDescriptor( )
cpp YFUN_DESCR get_functionDescriptor( )
m -(YFUN_DESCR) functionDescriptor
pas function get_functionDescriptor( ): YFUN_DESCR
vb function get_functionDescriptor( ) As YFUN_DESCR
cs YFUN_DESCR get_functionDescriptor( )
java String get_functionDescriptor( )
py def get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

## qt→get\_functionId() qt→functionId()qt.get\_functionId()

YQt

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

js	function <b>get_functionId( )</b>
nodejs	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

Par exemple `relay1`.

### Retourne :

une chaîne de caractères identifiant l'élément de quaternion (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**qt→get\_hardwareId()  
qt→hardwareId()qt.get\_hardwareId()****YQt**

Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL.FUNCTIONID.

```
js function get_hardwareId( )  
node.js function get_hardwareId( )  
php function get_hardwareId( )  
cpp string get_hardwareId( )  
m -(NSString*) hardwareId  
vb function get_hardwareId( ) As String  
cs string get_hardwareId( )  
java String get_hardwareId( )  
py def get_hardwareId( )
```

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() qt→highestValue()qt.get\_highestValue()

YQt

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YSensor target <b>get_highestValue</b>

### 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.

```
js function get_logFrequency( )  
nodejs function get_logFrequency( )  
php function get_logFrequency( )  
cpp string get_logFrequency( )  
m -(NSString*) logFrequency  
pas function get_logFrequency( ):string  
vb function get_logFrequency( ) As String  
cs string get_logFrequency( )  
java String get_logFrequency( )  
py def get_logFrequency( )  
cmd YSensor target 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() qt→logicalName()qt.get\_logicalName()

YQt

Retourne le nom logique de l'élément de quaternion.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YSensor target 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()**  
**qt→lowestValue()qt.get\_lowestValue()****YQt**

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

```
js function get_lowestValue( )  
node.js function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YSensor target 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()**  
**qt→module()qt.get\_module()**

YQt

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module( )</b>
nodejs	function <b>get_module( )</b>
php	function <b>get_module( )</b>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module( ): TYModule</b>
vb	function <b>get_module( ) As YModule</b>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

## qt→get\_module\_async() qt→module\_async()

YQt

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
nodejs 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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YSensor target 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.

```
js function get_reportFrequency( )  
nodejs function get_reportFrequency( )  
php function get_reportFrequency( )  
cpp string get_reportFrequency( )  
m -(NSString*) reportFrequency  
pas function get_reportFrequency( ): string  
vb function get_reportFrequency( ) As String  
cs string get_reportFrequency( )  
java String get_reportFrequency( )  
py def get_reportFrequency( )  
cmd YSensor target get_reportFrequency
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

## qt→get\_resolution() qt→resolution()qt.get\_resolution()

YQt

Retourne la résolution des valeurs mesurées.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YSensor target 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.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	YSensor <b>target get_unit</b>

**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) qt→userData()qt.get(userData)

YQt

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
js function get(userData) 
nodejs function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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.

```
js function isOnline( )  
nodejs function isOnline( )  
php function isOnline( )  
cpp bool isOnline( )  
m -(BOOL) isOnline  
pas function isOnline( ): boolean  
vb function isOnline( ) As Boolean  
cs bool isOnline( )  
java boolean isOnline( )  
py def isOnline( )
```

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()

YQt

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)

cmd  YSensor target loadCalibrationPoints rawValues refValues

```

**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()

YQt

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextQt()</b>
nodejs	function <b>nextQt()</b>
php	function <b>nextQt()</b>
cpp	YQt * <b>nextQt()</b>
m	-(YQt*) <b>nextQt</b>
pas	function <b>nextQt()</b> : TYQt
vb	function <b>nextQt()</b> As YQt
cs	YQt <b>nextQt()</b>
java	YQt <b>nextQt()</b>
py	def <b>nextQt()</b>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YQtTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YQtTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYQtTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

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.

```
js function registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YQtValueCallback callback)
m -(int) registerValueCallback : (YQtValueCallback) callback
pas function registerValueCallback( callback: TYQtValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**qt→set\_highestValue()  
qt→setHighestValue()qt.set\_highestValue()**

YQt

Modifie la mémoire de valeur maximale observée.

```
js function set_highestValue( newval)
node.js function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YSensor target set_highestValue newval
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logFrequency()**

YQt

**qt→setLogFrequency()qt.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

<b>js</b>	function <b>set_logFrequency( newval)</b>
<b>node.js</b>	function <b>set_logFrequency( newval)</b>
<b>php</b>	function <b>set_logFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_logFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogFrequency : (NSString*) newval</b>
<b>pas</b>	function <b>set_logFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logFrequency( string newval)</b>
<b>java</b>	int <b>set_logFrequency( String newval)</b>
<b>py</b>	def <b>set_logFrequency( newval)</b>
<b>cmd</b>	<b>YSensor target set_logFrequency newval</b>

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logicalName()**  
**qt→setLogicalName()qt.set\_logicalName()**

YQt

Modifie le nom logique de l'élément de quaternion.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YSensor 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 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.

```
js function set_lowestValue( newval)
nodejs function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YSensor target set_lowestValue newval
```

### 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.

```
js function set_reportFrequency( newval)
node.js function set_reportFrequency( newval)
php function set_reportFrequency( $newval)
cpp int set_reportFrequency( const string& newval)
m -(int) setReportFrequency : (NSString*) newval
pas function set_reportFrequency( newval: string): integer
vb function set_reportFrequency( ByVal newval As String) As Integer
cs int set_reportFrequency( string newval)
java int set_reportFrequency( String newval)
py def set_reportFrequency( newval)
cmd YSensor target 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() qt→setResolution()qt.set\_resolution()

YQt

Modifie la résolution des valeurs physique mesurées.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YSensor target set_resolution newval
```

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).

```
js function set(userData) data
nodejs function set(userData) data
php function set(userData) $data
cpp void set(userData) void* data
m -(void) setUserData : (void*) data
pas procedure set(userData) data: Tobject
vb procedure set(userData) ByVal data As Object
cs void set(userData) object data
java void set(userData) Object data
py def set(userData) data
```

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()****YQt**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)  
nodejs 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()****YRealTimeClock****yFindRealTimeClock()****YRealTimeClock.FindRealTimeClock()**

Permet de retrouver une horloge d'après un identifiant donné.

js	function <b>yFindRealTimeClock( func)</b>
nodejs	function <b>FindRealTimeClock( func)</b>
php	function <b>yFindRealTimeClock( \$func)</b>
cpp	<b>YRealTimeClock*</b> <b>yFindRealTimeClock( const string&amp; func)</b>
m	<b>YRealTimeClock*</b> <b>yFindRealTimeClock( NSString* func)</b>
pas	function <b>yFindRealTimeClock( func: string): TYRealTimeClock</b>
vb	function <b>yFindRealTimeClock( ByVal func As String) As YRealTimeClock</b>
cs	<b>YRealTimeClock</b> <b>FindRealTimeClock( string func)</b>
java	<b>YRealTimeClock</b> <b>FindRealTimeClock( String func)</b>
py	def <b>FindRealTimeClock( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnLine()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'horloge sans ambiguïté

**Retourne :**

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

**YRealTimeClock.FirstRealTimeClock()****YRealTimeClock****yFirstRealTimeClock()****YRealTimeClock.FirstRealTimeClock()**

Commence l'énumération des horloge accessibles par la librairie.

js	function <b>yFirstRealTimeClock( )</b>
node.js	function <b>FirstRealTimeClock( )</b>
php	function <b>yFirstRealTimeClock( )</b>
cpp	YRealTimeClock* <b>yFirstRealTimeClock( )</b>
m	YRealTimeClock* <b>yFirstRealTimeClock( )</b>
pas	function <b>yFirstRealTimeClock( )</b> : TYRealTimeClock
vb	function <b>yFirstRealTimeClock( )</b> As YRealTimeClock
cs	YRealTimeClock <b>FirstRealTimeClock( )</b>
java	YRealTimeClock <b>FirstRealTimeClock( )</b>
py	def <b>FirstRealTimeClock( )</b>

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.

<b>js</b>	<b>function describe( )</b>
<b>nodejs</b>	<b>function describe( )</b>
<b>php</b>	<b>function describe( )</b>
<b>cpp</b>	<b>string describe( )</b>
<b>m</b>	<b>-(NSString*) describe</b>
<b>pas</b>	<b>function describe( ): string</b>
<b>vb</b>	<b>function describe( ) As String</b>
<b>cs</b>	<b>string describe( )</b>
<b>java</b>	<b>String describe( )</b>
<b>py</b>	<b>def describe( )</b>

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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'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).

```
js    function get_advertisedValue( )  
node.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   YRealTimeClock target 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"

js	function getDateTime( )
nodejs	function getDateTime( )
php	function getDateTime( )
cpp	string getDateTime( )
m	-(NSString*) dateTime
pas	function getDateTime( ): string
vb	function getDateTime( ) As String
cs	string getDateTime( )
java	String getDateTime( )
py	def 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.

js	function <b>get_errorMessage( )</b>
node.js	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	String <b>get_errorMessage( )</b>
py	def <b>get_errorMessage( )</b>

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.

js	function get_errorType( )
nodejs	function get_errorType( )
php	function get_errorType( )
cpp	YRETCODE get_errorType( )
pas	function get_errorType( ): YRETCODE
vb	function get_errorType( ) As YRETCODE
cs	YRETCODE get_errorType( )
java	int get_errorType( )
py	def get_errorType( )

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

js	function get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

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.

```
js    function get_logicalName( )  
nodejs function get_logicalName( )  
php   function get_logicalName( )  
cpp   string get_logicalName( )  
m     -(NSString*) logicalName  
pas   function get_logicalName( ): string  
vb    function get_logicalName( ) As String  
cs    string get_logicalName( )  
java  String get_logicalName( )  
py    def get_logicalName( )  
cmd   YRealTimeClock target get_logicalName
```

**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.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YRealTimeClock**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

js	function <b>get_timeSet( )</b>
node.js	function <b>get_timeSet( )</b>
php	function <b>get_timeSet( )</b>
cpp	Y_TIMESET_enum <b>get_timeSet( )</b>
m	-(Y_TIMESET_enum) timeSet
pas	function <b>get_timeSet( )</b> : Integer
vb	function <b>get_timeSet( )</b> As Integer
cs	int <b>get_timeSet( )</b>
java	int <b>get_timeSet( )</b>
py	def <b>get_timeSet( )</b>
cmd	YRealTimeClock <b>target get_timeSet</b>

**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).

js	function <b>get_unixTime( )</b>
nodejs	function <b>get_unixTime( )</b>
php	function <b>get_unixTime( )</b>
cpp	s64 <b>get_unixTime( )</b>
m	-(s64) unixTime
pas	function <b>get_unixTime( )</b> : int64
vb	function <b>get_unixTime( )</b> As Long
cs	long <b>get_unixTime( )</b>
java	long <b>get_unixTime( )</b>
py	def <b>get_unixTime( )</b>
cmd	YRealTimeClock <b>target get_unixTime</b>

**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).

```
js function get(userData) 
nodejs function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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).

js	function <b>get_utcOffset()</b>
node.js	function <b>get_utcOffset()</b>
php	function <b>get_utcOffset()</b>
cpp	int <b>get_utcOffset()</b>
m	-(int) <b>utcOffset</b>
pas	function <b>get_utcOffset()</b> : LongInt
vb	function <b>get_utcOffset()</b> As Integer
cs	int <b>get_utcOffset()</b>
java	int <b>get_utcOffset()</b>
py	def <b>get_utcOffset()</b>
cmd	YRealTimeClock <b>target get_utcOffset</b>

**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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()****YRealTimeClock**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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()

## YRealTimeClock

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextRealTimeClock( )</b>
node.js	function <b>nextRealTimeClock( )</b>
php	function <b>nextRealTimeClock( )</b>
cpp	YRealTimeClock * <b>nextRealTimeClock( )</b>
m	-{YRealTimeClock*} <b>nextRealTimeClock</b>
pas	function <b>nextRealTimeClock( ): TYRealTimeClock</b>
vb	function <b>nextRealTimeClock( ) As YRealTimeClock</b>
cs	YRealTimeClock <b>nextRealTimeClock( )</b>
java	YRealTimeClock <b>nextRealTimeClock( )</b>
py	def <b>nextRealTimeClock( )</b>

**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.

```
js function registerValueCallback( callback)
nodejs function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YRealTimeClockValueCallback callback)
m -(int) registerValueCallback : (YRealTimeClockValueCallback) callback
pas function registerValueCallback( callback: TYRealTimeClockValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**realtimeclock→set\_logicalName()**  
**realtimeclock→setLogicalName()**  
**realtimeclock.set\_logicalName()**

**YRealTimeClock**

Modifie le nom logique de l'horloge.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YRealTimeClock 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 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.

<b>js</b>	function <b>set_unixTime( newval)</b>
<b>node.js</b>	function <b>set_unixTime( newval)</b>
<b>php</b>	function <b>set_unixTime( \$newval)</b>
<b>cpp</b>	int <b>set_unixTime( s64 newval)</b>
<b>m</b>	-(int) <b>setUnixTime : (s64) newval</b>
<b>pas</b>	function <b>set_unixTime( newval: int64): integer</b>
<b>vb</b>	function <b>set_unixTime( ByVal newval As Long) As Integer</b>
<b>cs</b>	int <b>set_unixTime( long newval)</b>
<b>java</b>	int <b>set_unixTime( long newval)</b>
<b>py</b>	def <b>set_unixTime( newval)</b>
<b>cmd</b>	<b>YRealTimeClock target set_unixTime newval</b>

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).

```
js function set(userData) {  
nodejs function set(userData) {  
php function set(userData) {  
cpp void set(userData) {  
m -(void) setUserData : (void*) data  
pas procedure set(userData: Tobject)  
vb procedure set(userData( ByVal data As Object)  
cs void set(userData( object data  
java void set(userData( Object data  
py def set(userData( data)
```

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).

js	function <b>set_utcOffset( newval)</b>
node.js	function <b>set_utcOffset( newval)</b>
php	function <b>set_utcOffset( \$newval)</b>
cpp	int <b>set_utcOffset( int newval)</b>
m	-(int) setUtcOffset : (int) <b>newval</b>
pas	function <b>set_utcOffset( newval: LongInt): integer</b>
vb	function <b>set_utcOffset( ByVal newval As Integer) As Integer</b>
cs	int <b>set_utcOffset( int newval)</b>
java	int <b>set_utcOffset( int newval)</b>
py	def <b>set_utcOffset( newval)</b>
cmd	<b>YRealTimeClock target set_utcOffset newval</b>

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()

## YRealTimeClock

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()****YRefFrame****yFindRefFrame() YRefFrame.FindRefFrame()**

Permet de retrouver un référentiel d'après un identifiant donné.

<b>js</b>	<code>function yFindRefFrame( func)</code>
<b>node.js</b>	<code>function FindRefFrame( func)</code>
<b>php</b>	<code>function yFindRefFrame( \$func)</code>
<b>cpp</b>	<code>YRefFrame* yFindRefFrame( const string&amp; func)</code>
<b>m</b>	<code>YRefFrame* yFindRefFrame( NSString* func)</code>
<b>pas</b>	<code>function yFindRefFrame( func: string): TYRefFrame</code>
<b>vb</b>	<code>function yFindRefFrame( ByVal func As String) As YRefFrame</code>
<b>cs</b>	<code>YRefFrame FindRefFrame( string func)</code>
<b>java</b>	<code>YRefFrame FindRefFrame( String func)</code>
<b>py</b>	<code>def FindRefFrame( func)</code>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRefFrame.isOnline()` pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le référentiel sans ambiguïté

**Retourne :**

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

**YRefFrame.FirstRefFrame()****YRefFrame****yFirstRefFrame()YRefFrame.FirstRefFrame()**

Commence l'énumération des référentiels accessibles par la librairie.

js	function <b>yFirstRefFrame( )</b>
nodejs	function <b>FirstRefFrame( )</b>
php	function <b>yFirstRefFrame( )</b>
cpp	YRefFrame* <b>yFirstRefFrame( )</b>
m	YRefFrame* <b>yFirstRefFrame( )</b>
pas	function <b>yFirstRefFrame( ): TYRefFrame</b>
vb	function <b>yFirstRefFrame( ) As YRefFrame</b>
cs	YRefFrame <b>FirstRefFrame( )</b>
java	YRefFrame <b>FirstRefFrame( )</b>
py	def <b>FirstRefFrame( )</b>

Utiliser la fonction `YRefFrame.nextRefFrame( )` pour itérer sur les autres référentiels.

**Retourne :**

un pointeur sur un objet `YRefFrame`, correspondant à le 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.

```
js function cancel3DCalibration( )  
node.js function cancel3DCalibration( )  
php function cancel3DCalibration( )  
cpp int cancel3DCalibration( )  
m -(int) cancel3DCalibration  
pas function cancel3DCalibration( ): LongInt  
vb function cancel3DCalibration( ) As Integer  
cs int cancel3DCalibration( )  
java int cancel3DCalibration( )  
py def cancel3DCalibration( )  
cmd YRefFrame target 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le référentiel (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

`refframe→get_3DCalibrationHint()`  
`refframe→3DCalibrationHint()`  
`refframe.get_3DCalibrationHint()`

**YRefFrame**

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

```
js function get_3DCalibrationHint( )  
nodejs function get_3DCalibrationHint( )  
php function get_3DCalibrationHint( )  
cpp string get_3DCalibrationHint( )  
m -(NSString*) 3DCalibrationHint  
pas function get_3DCalibrationHint( ): string  
vb function get_3DCalibrationHint( ) As String  
cs string get_3DCalibrationHint( )  
java String get_3DCalibrationHint( )  
py def get_3DCalibrationHint( )  
cmd YRefFrame target 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.

<b>js</b>	function <b>get_3DCalibrationLogMsg( )</b>
<b>node.js</b>	function <b>get_3DCalibrationLogMsg( )</b>
<b>php</b>	function <b>get_3DCalibrationLogMsg( )</b>
<b>cpp</b>	string <b>get_3DCalibrationLogMsg( )</b>
<b>m</b>	-(NSString*) <b>3DCalibrationLogMsg</b>
<b>pas</b>	function <b>get_3DCalibrationLogMsg( )</b> : string
<b>vb</b>	function <b>get_3DCalibrationLogMsg( ) As String</b>
<b>cs</b>	string <b>get_3DCalibrationLogMsg( )</b>
<b>java</b>	String <b>get_3DCalibrationLogMsg( )</b>
<b>py</b>	def <b>get_3DCalibrationLogMsg( )</b>
<b>cmd</b>	<b>YRefFrame target get_3DCalibrationLogMsg</b>

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`.

```
js function get_3DCalibrationProgress( )  
nodejs function get_3DCalibrationProgress( )  
php function get_3DCalibrationProgress( )  
cpp int get_3DCalibrationProgress( )  
m -(int) 3DCalibrationProgress  
pas function get_3DCalibrationProgress( ): LongInt  
vb function get_3DCalibrationProgress( ) As Integer  
cs int get_3DCalibrationProgress( )  
java int get_3DCalibrationProgress( )  
py def get_3DCalibrationProgress( )  
cmd YRefFrame target 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.

```
js function get_3DCalibrationStage( )  
nodejs function get_3DCalibrationStage( )  
php function get_3DCalibrationStage( )  
cpp int get_3DCalibrationStage( )  
m -(int) 3DCalibrationStage  
pas function get_3DCalibrationStage( ): LongInt  
vb function get_3DCalibrationStage( ) As Integer  
cs int get_3DCalibrationStage( )  
java int get_3DCalibrationStage( )  
py def get_3DCalibrationStage( )  
cmd YRefFrame target get_3DCalibrationStage
```

**Retourne :**

une nombre entier, croissant au fur et à mesure de la complétion des étapes.

`refframe→get_3DCalibrationStageProgress()`  
`refframe→3DCalibrationStageProgress()`  
`refframe.get_3DCalibrationStageProgress()`

**YRefFrame**

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

```
js function get_3DCalibrationStageProgress( )  
nodejs function get_3DCalibrationStageProgress( )  
php function get_3DCalibrationStageProgress( )  
cpp int get_3DCalibrationStageProgress( )  
m -(int) 3DCalibrationStageProgress  
pas function get_3DCalibrationStageProgress( ): LongInt  
vb function get_3DCalibrationStageProgress( ) As Integer  
cs int get_3DCalibrationStageProgress( )  
java int get_3DCalibrationStageProgress( )  
py def get_3DCalibrationStageProgress( )  
cmd YRefFrame target get_3DCalibrationStageProgress
```

**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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YRefFrame <b>target get_advertisedValue</b>

**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.

```
js function get_bearing( )  
node.js function get_bearing( )  
php function get_bearing( )  
cpp double get_bearing( )  
m -(double) bearing  
pas function get_bearing( ): double  
vb function get_bearing( ) As Double  
cs double get_bearing( )  
java double get_bearing( )  
py def get_bearing( )  
cmd YRefFrame target get_bearing
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici.

**Retourne :**

une valeur numérique représentant le cap de référence utilisé par le compas

En cas d'erreur, déclenche une exception ou retourne `Y_BEARING_INVALID`.

**refframe→get\_errorMessage()  
refframe→errorMessage()  
refframe.get\_errorMessage()****YRefFrame**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

js	function <b>get_errorMessage( )</b>
node.js	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	String <b>get_errorMessage( )</b>
py	<b>def get_errorMessage( )</b>

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

```
js function get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

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.

js	function <b>get_functionDescriptor()</b>
nodejs	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) <b>functionDescriptor</b>
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

js	function get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du référentiel (par exemple RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**refframe→get\_logicalName()****YRefFrame****refframe→logicalName()refframe.get\_logicalName()**

Retourne le nom logique du référentiel.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YRefFrame target 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()**  
**refframe→module()refframe.get\_module()****YRefFrame**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YRefFrame**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

**refframe→get\_mountOrientation()****YRefFrame****refframe→mountOrientation()****refframe.get\_mountOrientation()**

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

```
js function get_mountOrientation( )
nodejs function get_mountOrientation( )
php function get_mountOrientation( )
cpp Y_MOUNTORIENTATION get_mountOrientation( )
m -(Y_MOUNTORIENTATION) mountOrientation
pas function get_mountOrientation( ): TYMOUNTORIENTATION
vb function get_mountOrientation( ) As Y_MOUNTORIENTATION
cs MOUNTORIENTATION get_mountOrientation( )
java MOUNTORIENTATION get_mountOrientation( )
py def get_mountOrientation( )
cmd YRefFrame target get_mountOrientation
```

**Retourne :**

une valeur parmi l'énumération Y\_MOUNTORIENTATION (Y\_MOUNTORIENTATION\_TWELVE, Y\_MOUNTORIENTATION\_THREE, Y\_MOUNTORIENTATION\_SIX, Y\_MOUNTORIENTATION\_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get\_mountPosition()**  
**refframe→mountPosition()**  
**refframe.get\_mountPosition()**

**YRefFrame**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

js	function <b>get_mountPosition( )</b>
node.js	function <b>get_mountPosition( )</b>
php	function <b>get_mountPosition( )</b>
cpp	Y_MOUNTPOSITION <b>get_mountPosition( )</b>
m	-(Y_MOUNTPOSITION) mountPosition
pas	function <b>get_mountPosition( )</b> : TYMOUNTPOSITION
vb	function <b>get_mountPosition( )</b> As Y_MOUNTPOSITION
cs	MOUNTPOSITION <b>get_mountPosition( )</b>
java	MOUNTPOSITION <b>get_mountPosition( )</b>
py	def <b>get_mountPosition( )</b>
cmd	YRefFrame <b>target get_mountPosition</b>

**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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData) {  
m -(void*) userData  
pas function get(userData): Object  
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.

**refframe→isOnline()|refframe.isOnline()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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.

```
js   function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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.

**refframe→more3DCalibration()**  
**refframe.more3DCalibration()****YRefFrame**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.

js	function more3DCalibration( )
node.js	function more3DCalibration( )
php	function more3DCalibration( )
cpp	int more3DCalibration( )
m	-(int) more3DCalibration
pas	function more3DCalibration( ): LongInt
vb	function more3DCalibration( ) As Integer
cs	int more3DCalibration( )
java	int more3DCalibration( )
py	def more3DCalibration( )
cmd	YRefFrame target more3DCalibration

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()`.

js	function <b>nextRefFrame()</b>
nodejs	function <b>nextRefFrame()</b>
php	function <b>nextRefFrame()</b>
cpp	<b>YRefFrame * nextRefFrame()</b>
m	<b>-(YRefFrame*) nextRefFrame</b>
pas	function <b>nextRefFrame()</b> : TYRefFrame
vb	function <b>nextRefFrame()</b> As YRefFrame
cs	<b>YRefFrame nextRefFrame()</b>
java	<b>YRefFrame nextRefFrame()</b>
py	<b>def nextRefFrame()</b>

**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.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
node.js	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YRefFrameValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YRefFrameValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYRefFrameValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**refframe→save3DCalibration()**  
**refframe.save3DCalibration()****YRefFrame**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

```
js function save3DCalibration( )
node.js function save3DCalibration( )
php function save3DCalibration( )
cpp int save3DCalibration( )
m -(int) save3DCalibration
pas function save3DCalibration( ): LongInt
vb function save3DCalibration( ) As Integer
cs int save3DCalibration( )
java int save3DCalibration( )
py def save3DCalibration( )
cmd YRefFrame target save3DCalibration
```

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.

<b>js</b>	<code>function set_bearing( newval)</code>
<b>nodejs</b>	<code>function set_bearing( newval)</code>
<b>php</b>	<code>function set_bearing( \$newval)</code>
<b>cpp</b>	<code>int set_bearing( double newval)</code>
<b>m</b>	<code>-(int) setBearing : (double) newval</code>
<b>pas</b>	<code>function set_bearing( newval: double): integer</code>
<b>vb</b>	<code>function set_bearing( ByVal newval As Double) As Integer</code>
<b>cs</b>	<code>int set_bearing( double newval)</code>
<b>java</b>	<code>int set_bearing( double newval)</code>
<b>py</b>	<code>def set_bearing( newval)</code>
<b>cmd</b>	<code>YRefFrame target set_bearing newval</code>

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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YRefFrame 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 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.

```

js   function set_mountPosition( position, orientation)
nodejs function set_mountPosition( position, orientation)
php  function set_mountPosition( $position, $orientation)
cpp   int set_mountPosition( Y_MOUNTPOSITION position,
                           Y_MOUNTORIENTATION orientation)
m    -(int) setMountPosition : (Y_MOUNTPOSITION) position
                           : (Y_MOUNTORIENTATION) orientation
pas   function set_mountPosition( position: TYMOUNTPOSITION,
                                 orientation: TYMOUNTORIENTATION): LongInt
vb    function set_mountPosition( ) As Integer
cs    int set_mountPosition( MOUNTPOSITION position,
                           MOUNTORIENTATION orientation)
java  int set_mountPosition( MOUNTPOSITION position,
                           MOUNTORIENTATION orientation)
py    def set_mountPosition( position, orientation)
cmd   YRefFrame target set_mountPosition position orientation

```

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).

```
js function set(userData) data
nodejs function set(userData) data
php function set(userData) $data
cpp void set(userData) void* data
m -(void) setUserData : (void*) data
pas procedure set(userData) data: Tobject
vb procedure set(userData) ByVal data As Object
cs void set(userData) object data
java void set(userData) Object data
py def set(userData) data
```

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.

```
js function start3DCalibration( )  
nodejs function start3DCalibration( )  
php function start3DCalibration( )  
cpp int start3DCalibration( )  
m -(int) start3DCalibration  
pas function start3DCalibration( ): LongInt  
vb function start3DCalibration( ) As Integer  
cs int start3DCalibration( )  
java int start3DCalibration( )  
py def start3DCalibration( )  
cmd YRefFrame target start3DCalibration
```

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()****YRefFrame**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()YRelay.FindRelay()

YRelay

Permet de retrouver un relais d'après un identifiant donné.

```
js function yFindRelay( func)
node.js function FindRelay( func)
php function yFindRelay( $func)
cpp YRelay* yFindRelay( const string& func)
m YRelay* yFindRelay( NSString* func)
pas function yFindRelay( func: string): TYRelay
vb function yFindRelay( ByVal func As String) As YRelay
cs YRelay FindRelay( string func)
java YRelay FindRelay( String func)
def FindRelay( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le relais soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRelay.isOnline()` pour tester si le relais est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le relais sans ambiguïté

### Retourne :

un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

## YRelay.FirstRelay()

## YRelay

### yFirstRelay() YRelay.FirstRelay()

Commence l'énumération des relais accessibles par la librairie.

```
js function yFirstRelay( )
nodejs function FirstRelay( )
php function yFirstRelay( )
cpp YRelay* yFirstRelay( )
m YRelay* yFirstRelay( )
pas function yFirstRelay( ): TYRelay
vb function yFirstRelay( ) As YRelay
cs YRelay FirstRelay( )
java YRelay FirstRelay( )
py def FirstRelay( )
```

Utiliser la fonction `YRelay.nextRelay()` pour itérer sur les autres relais.

#### Retourne :

un pointeur sur un objet `YRelay`, correspondant à le premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

**relay→delayedPulse()relay.delayedPulse()**

YRelay

Pré-programme une impulsion

```
js function delayedPulse( ms_delay, ms_duration)
nodejs function delayedPulse( ms_delay, ms_duration)
php function delayedPulse( $ms_delay, $ms_duration)
cpp int delayedPulse( int ms_delay, int ms_duration)
m -(int) delayedPulse : (int) ms_delay : (int) ms_duration
pas function delayedPulse( ms_delay: LongInt, ms_duration: LongInt): integer
vb function delayedPulse( ByVal ms_delay As Integer,
                           ByVal ms_duration As Integer) As Integer
cs int delayedPulse( int ms_delay, int ms_duration)
java int delayedPulse( int ms_delay, int ms_duration)
py def delayedPulse( ms_delay, ms_duration)
cmd YRelay target delayedPulse ms_delay ms_duration
```

**Paramètres :****ms\_delay** delai 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

```
js function get_advertisedValue( )  
node.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 YRelay target 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().

```
js function get_countdown( )
nodejs function get_countdown( )
php function get_countdown( )
cpp s64 get_countdown( )
m -(s64) countdown
pas function get_countdown( ): int64
vb function get_countdown( ) As Long
cs long get_countdown( )
java long get_countdown( )
py def get_countdown( )
cmd YRelay target get_countdown
```

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne Y\_COUNTDOWN\_INVALID.

## relay→get\_errorMessage() relay→errorMessage()relay.get\_errorMessage()

YRelay

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
js function get_errorMessage( )
node.js function get_errorMessage( )
php function get_errorMessage( )
cpp string get_errorMessage( )
m -(NSString*) errorMessage
pas function get_errorMessage( ): string
vb function get_errorMessage( ) As String
cs string get_errorMessage( )
java String get_errorMessage( )
py def get_errorMessage( )
```

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() relay→errorType()relay.get\_errorType()

YRelay

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

js	function get_errorType( )
nodejs	function get_errorType( )
php	function get_errorType( )
cpp	YRETCODE get_errorType( )
pas	function get_errorType( ): YRETCODE
vb	function get_errorType( ) As YRETCODE
cs	YRETCODE get_errorType( )
java	int get_errorType( )
py	def get_errorType( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du relais.

## relay→get\_friendlyName() YRelay relay→friendlyName()relay.get\_friendlyName()

Retourne un identifiant global du relais au format NOM\_MODULE.NOM\_FONCTION.

```
js function get_friendlyName( )  
node.js function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

Le chaîne renvoyée 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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	YFUN_DESCR <b>get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	YFUN_DESCR <b>get_functionDescriptor()</b>
java	String <b>get_functionDescriptor()</b>
py	def <b>get_functionDescriptor()</b>

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()  
relay→functionId()relay.get\_functionId()****YRelay**

Retourne l'identifiant matériel du relais, sans référence au module.

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le relais (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**relay→get\_hardwareId()****YRelay****relay→hardwareId()relay.get\_hardwareId()**

Retourne l'identifiant matériel unique du relais au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId()</b>
nodejs	function <b>get_hardwareId()</b>
php	function <b>get_hardwareId()</b>
cpp	string <b>get_hardwareId()</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId()</b> As String
cs	string <b>get_hardwareId()</b>
java	String <b>get_hardwareId()</b>
py	def <b>get_hardwareId()</b>

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.

```
js function get_logicalName( )
node.js function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YRelay target 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.

```
js function get_maxTimeOnStateA( )
nodejs function get_maxTimeOnStateA( )
php function get_maxTimeOnStateA( )
cpp s64 get_maxTimeOnStateA( )
m -(s64) maxTimeOnStateA
pas function get_maxTimeOnStateA( ): int64
vb function get_maxTimeOnStateA( ) As Long
cs long get_maxTimeOnStateA( )
java long get_maxTimeOnStateA( )
py def get_maxTimeOnStateA( )
cmd YRelay target 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.

```
js function get_maxTimeOnStateB( )
nodejs function get_maxTimeOnStateB( )
php function get_maxTimeOnStateB( )
cpp s64 get_maxTimeOnStateB( )
m -(s64) maxTimeOnStateB
pas function get_maxTimeOnStateB( ): int64
vb function get_maxTimeOnStateB( ) As Long
cs long get_maxTimeOnStateB( )
java long get_maxTimeOnStateB( )
py def get_maxTimeOnStateB( )
cmd YRelay target 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() relay→module()relay.get\_module()

YRelay

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module()</b>
nodejs	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	YModule * <b>get_module()</b>
m	-(YModule*) module
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	YModule <b>get_module()</b>
java	YModule <b>get_module()</b>
py	def <b>get_module()</b>

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule rentrée ne sera pas joignable.

**Retourne :**

une instance de YModule

## relay→get\_module\_async() relay→module\_async()

YRelay

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context )
node.js 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.

```
js function get_output( )
nodejs function get_output( )
php function get_output( )
cpp Y_OUTPUT_enum get_output( )
m -(Y_OUTPUT_enum) output
pas function get_output( ): Integer
vb function get_output( ) As Integer
cs int get_output( )
java int get_output( )
py def get_output( )
cmd YRelay target get_output
```

**Retourne :**

soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUT\_INVALID.

## relay→get\_pulseTimer() relay→pulseTimer()relay.get\_pulseTimer()

YRelay

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
js function get_pulseTimer( )  
nodejs function get_pulseTimer( )  
php function get_pulseTimer( )  
cpp s64 get_pulseTimer( )  
m -(s64) pulseTimer  
pas function get_pulseTimer( ): int64  
vb function get_pulseTimer( ) As Long  
cs long get_pulseTimer( )  
java long get_pulseTimer( )  
py def get_pulseTimer( )  
cmd YRelay target 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).

js	function <b>get_state( )</b>
nodejs	function <b>get_state( )</b>
php	function <b>get_state( )</b>
cpp	Y_STATE_enum <b>get_state( )</b>
m	-(Y_STATE_enum) state
pas	function <b>get_state( )</b> : Integer
vb	function <b>get_state( )</b> As Integer
cs	int <b>get_state( )</b>
java	int <b>get_state( )</b>
py	def <b>get_state( )</b>
cmd	YRelay target <b>get_state</b>

**Retourne :**

soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y\_STATE\_INVALID.

**relay→get\_stateAtPowerOn()** YRelay  
**relay→stateAtPowerOn()relay.get\_stateAtPowerOn()**

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
js function get_stateAtPowerOn( )  
nodejs function get_stateAtPowerOn( )  
php function get_stateAtPowerOn( )  
cpp Y_STATEATPOWERON_enum get_stateAtPowerOn( )  
m -(Y_STATEATPOWERON_enum) stateAtPowerOn  
pas function get_stateAtPowerOn( ): Integer  
vb function get_stateAtPowerOn( ) As Integer  
cs int get_stateAtPowerOn( )  
java int get_stateAtPowerOn( )  
py def get_stateAtPowerOn( )  
cmd YRelay target 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).

js	function <b>get(userData)</b> {
nodejs	function <b>get(userData)</b> {
php	function <b>get(userData)</b> {
cpp	void * <b>get(userData)</b> {
m	-(void*) userData;
pas	function <b>get(userData)</b> : Tobject {
vb	function <b>get(userData)</b> As Object {
cs	object <b>get(userData)</b> {
java	Object <b>get(userData)</b> {
py	def <b>get(userData)</b> {

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.

```
js function isOnline( )
nodejs function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

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()

## YRelay

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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()

## YRelay

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
js   function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextRelay( )</b>
nodejs	function <b>nextRelay( )</b>
php	function <b>nextRelay( )</b>
cpp	<b>YRelay * nextRelay( )</b>
m	<b>-(YRelay*) nextRelay</b>
pas	function <b>nextRelay( ): TYRelay</b>
vb	function <b>nextRelay( ) As YRelay</b>
cs	<b>YRelay nextRelay( )</b>
java	<b>YRelay nextRelay( )</b>
py	<b>def nextRelay( )</b>

**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).

```
js function pulse( ms_duration)
nodejs function pulse( ms_duration)
php function pulse( $ms_duration)
cpp int pulse( int ms_duration)
m -(int) pulse : (int) ms_duration
pas function pulse( ms_duration: LongInt): integer
vb function pulse( ByVal ms_duration As Integer) As Integer
cs int pulse( int ms_duration)
java int pulse( int ms_duration)
py def pulse( ms_duration)
cmd YRelay target pulse ms_duration
```

### 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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YRelayValueCallback callback)
m -(int) registerValueCallback : (YRelayValueCallback) callback
pas function registerValueCallback( callback: TYRelayValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>nodejs</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) setLogicalName : (NSString*) <b>newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YRelay target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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.

```
js function set_maxTimeOnStateA( newval)
nodejs function set_maxTimeOnStateA( newval)
php function set_maxTimeOnStateA( $newval)
cpp int set_maxTimeOnStateA( s64 newval)
m -(int) setMaxTimeOnStateA : (s64) newval
pas function set_maxTimeOnStateA( newval: int64): integer
vb function set_maxTimeOnStateA( ByVal newval As Long) As Integer
cs int set_maxTimeOnStateA( long newval)
java int set_maxTimeOnStateA( long newval)
py def set_maxTimeOnStateA( newval)
cmd YRelay target 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.

js	function <b>set_maxTimeOnStateB( newval)</b>
node.js	function <b>setMaxTimeOnStateB( newval)</b>
php	function <b>set_maxTimeOnStateB( \$newval)</b>
cpp	int <b>set_maxTimeOnStateB( s64 newval)</b>
m	-(int) setMaxTimeOnStateB : (s64) newval
pas	function <b>set_maxTimeOnStateB( newval: int64): integer</b>
vb	function <b>set_maxTimeOnStateB( ByVal newval As Long) As Integer</b>
cs	int <b>set_maxTimeOnStateB( long newval)</b>
java	int <b>set_maxTimeOnStateB( long newval)</b>
py	def <b>set_maxTimeOnStateB( newval)</b>
cmd	YRelay <b>target set_maxTimeOnStateB newval</b>

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.

```
js function set_output( newval)
node.js function set_output( newval)
php function set_output( $newval)
cpp int set_output( Y_OUTPUT_enum newval)
m -(int) setOutput : (Y_OUTPUT_enum) newval
pas function set_output( newval: Integer): integer
vb function set_output( ByVal newval As Integer) As Integer
cs int set_output( int newval)
java int set_output( int newval)
py def set_output( newval)
cmd YRelay target 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).

js	function set_state( newval)
nodejs	function set_state( newval)
php	function set_state( \$newval)
cpp	int set_state( Y_STATE_enum newval)
m	-(int) setState : (Y_STATE_enum) newval
pas	function set_state( newval: Integer): integer
vb	function set_state( ByVal newval As Integer) As Integer
cs	int set_state( int newval)
java	int set_state( int newval)
py	def set_state( newval)
cmd	YRelay target set_state newval

**Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_stateAtPowerOn()**  
**relay→setStateAtPowerOn()**  
**relay.set\_stateAtPowerOn()**

YRelay

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
js function set_stateAtPowerOn( newval)
nodejs function set_stateAtPowerOn( newval)
php function set_stateAtPowerOn( $newval)
cpp int set_stateAtPowerOn( Y_STATEATPOWERON_enum newval)
m -(int) setStateAtPowerOn : (Y_STATEATPOWERON_enum) newval
pas function set_stateAtPowerOn( newval: Integer): integer
vb function set_stateAtPowerOn( ByVal newval As Integer) As Integer
cs int set_stateAtPowerOn( int newval)
java int set_stateAtPowerOn( int newval)
py def set_stateAtPowerOn( newval)
cmd YRelay target 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).

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	-(void) <b>setUserData : (void*) data</b>
pas	procedure <b>set(userData Tobject)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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()

YRelay

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()****YSensor****yFindSensor()YSensor.FindSensor()**

Permet de retrouver un senseur d'après un identifiant donné.

js	function <b>yFindSensor( func)</b>
node.js	function <b>FindSensor( func)</b>
php	function <b>yFindSensor( \$func)</b>
cpp	<b>YSensor*</b> <b>yFindSensor( string func)</b>
m	+( <b>YSensor*</b> ) <b>yFindSensor : (NSString*) func</b>
pas	function <b>yFindSensor( func: string): TYSensor</b>
vb	function <b>yFindSensor( ByVal func As String) As YSensor</b>
cs	<b>YSensor FindSensor( string func)</b>
java	<b>YSensor FindSensor( String func)</b>
py	def <b>FindSensor( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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()YSensor.FirstSensor()**

Commence l'énumération des senseurs accessibles par la librairie.

js	function <b>yFirstSensor()</b>
nodejs	function <b>FirstSensor()</b>
php	function <b>yFirstSensor()</b>
cpp	YSensor* <b>yFirstSensor()</b>
m	YSensor* <b>yFirstSensor()</b>
pas	function <b>yFirstSensor()</b> : TYSensor
vb	function <b>yFirstSensor()</b> As YSensor
cs	YSensor <b>FirstSensor()</b>
java	YSensor <b>FirstSensor()</b>
py	def <b>FirstSensor()</b>

Utiliser la fonction `YSensor.nextSensor()` pour itérer sur les autres senseurs.

**Retourne :**

un pointeur sur un objet `YSensor`, correspondant à le 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                           vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                           refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YSensor target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le senseur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**sensor→get\_advertisedValue()**  
**sensor→advertisedValue()**  
**sensor.get\_advertisedValue()**

YSensor

Retourne la valeur courante du senseur (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YSensor target get_advertisedValue
```

**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 rentrée par le capteur (sans arrondi ni calibration).

```
js    function get_currentRawValue( )  
node.js function get_currentRawValue( )  
php   function get_currentRawValue( )  
cpp   double get_currentRawValue( )  
m     -(double) currentRawValue  
pas   function get_currentRawValue( ): double  
vb    function get_currentRawValue( ) As Double  
cs    double get_currentRawValue( )  
java  double get_currentRawValue( )  
py    def get_currentRawValue( )  
cmd   YSensor target get_currentRawValue
```

**Retourne :**

une valeur numérique représentant la valeur brute rentrée 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.

```
js function get_currentValue( )
node.js function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YSensor target 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.

js	function <b>getErrorMessage( )</b>
node.js	function <b>getErrorMessage( )</b>
php	function <b>getErrorMessage( )</b>
cpp	string <b>getErrorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>getErrorMessage( )</b> : string
vb	function <b>getErrorMessage( )</b> As String
cs	string <b>getErrorMessage( )</b>
java	String <b>getErrorMessage( )</b>
py	def <b>getErrorMessage( )</b>

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()  
sensor→errorType()sensor.get\_errorType()****YSensor**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

**sensor→get\_friendlyName()****YSensor****sensor→friendlyName()sensor.get\_friendlyName()**

Retourne un identifiant global du senseur au format NOM\_MODULE.NOM\_FONCTION.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

---

<b>sensor→get_functionDescriptor()</b>	<b>YSensor</b>
<b>sensor→functionDescriptor()</b>	
<b>sensor.get_functionDescriptor()</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

<code>js</code>	<code>function get_functionDescriptor( )</code>
<code>nodejs</code>	<code>function get_functionDescriptor( )</code>
<code>php</code>	<code>function get_functionDescriptor( )</code>
<code>cpp</code>	<code>YFUN_DESCR get_functionDescriptor( )</code>
<code>m</code>	<code>-(YFUN_DESCR) functionDescriptor</code>
<code>pas</code>	<code>function get_functionDescriptor( ): YFUN_DESCR</code>
<code>vb</code>	<code>function get_functionDescriptor( ) As YFUN_DESCR</code>
<code>cs</code>	<code>YFUN_DESCR get_functionDescriptor( )</code>
<code>java</code>	<code>String get_functionDescriptor( )</code>
<code>py</code>	<code>def get_functionDescriptor( )</code>

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.

js	function <b>get_functionId( )</b>
nodejs	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**sensor→get\_hardwareId()****YSensor****sensor→hardwareId()sensor.get\_hardwareId()**

Retourne l'identifiant matériel unique du senseur au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du senseur (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**sensor→get\_highestValue()****YSensor****sensor→highestValue()sensor.get\_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( ): double</b>
vb	function <b>get_highestValue( ) As Double</b>
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	<b>YSensor target get_highestValue</b>

**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.

```
js function get_logFrequency( )  
nodejs function get_logFrequency( )  
php function get_logFrequency( )  
cpp string get_logFrequency( )  
m -(NSString*) logFrequency  
pas function get_logFrequency( ):string  
vb function get_logFrequency( ) As String  
cs string get_logFrequency( )  
java String get_logFrequency( )  
py def get_logFrequency( )  
cmd YSensor target 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()**  
**sensor→logicalName()sensor.get\_logicalName()****YSensor**

Retourne le nom logique du senseur.

js	function <b>get_logicalName( )</b>
nodejs	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	YSensor target <b>get_logicalName</b>

**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.

```
js function get_lowestValue( )  
node.js function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YSensor target 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()**  
**sensor→module()sensor.get\_module()****YSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module( )</b>
nodejs	function <b>get_module( )</b>
php	function <b>get_module( )</b>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module( ): TYModule</b>
vb	function <b>get_module( ) As YModule</b>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**sensor→get\_module\_async()**  
**sensor→module\_async()****YSensor**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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.

**sensor→get\_recordedData()**

# YSensor

**sensor→recordedData()|sensor.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
js function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php function get_recordedData( $startTime, $endTime)
cpp YDataSet get_recordedData( s64 startTime, s64 endTime)
m -(YDataSet*) recordedData : (s64) startTime
                           : (s64) endTime

pas function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb function get_recordedData( ) As YDataSet
cs YDataSet get_recordedData( long startTime, long endTime)
java YDataSet get_recordedData( long startTime, long endTime)
py def get_recordedData( startTime, endTime)
cmd YSensor target 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 find 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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YSensor target get_reportFrequency
```

**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()**  
**sensor→resolution()sensor.get\_resolution()****YSensor**

Retourne la résolution des valeurs mesurées.

js	function <b>get_resolution( )</b>
nodejs	function <b>get_resolution( )</b>
php	function <b>get_resolution( )</b>
cpp	double <b>get_resolution( )</b>
m	-(double) resolution
pas	function <b>get_resolution( ): double</b>
vb	function <b>get_resolution( ) As Double</b>
cs	double <b>get_resolution( )</b>
java	double <b>get_resolution( )</b>
py	def <b>get_resolution( )</b>
cmd	<b>YSensor target get_resolution</b>

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()**  
**sensor→unit()sensor.get\_unit()****YSensor**

Retourne l'unité dans laquelle la mesure est exprimée.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	YSensor <b>target get_unit</b>

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)

cmd  YSensor target loadCalibrationPoints rawValues refValues

```

### 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()

YSensor

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextSensor()</b>
nodejs	function <b>nextSensor()</b>
php	function <b>nextSensor()</b>
cpp	YSensor * <b>nextSensor()</b>
m	-( <b>YSensor*</b> ) <b>nextSensor</b>
pas	function <b>nextSensor()</b> : TYSensor
vb	function <b>nextSensor()</b> As YSensor
cs	YSensor <b>nextSensor()</b>
java	YSensor <b>nextSensor()</b>
py	def <b>nextSensor()</b>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YSensorTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YSensorTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYSensorTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
node.js	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YSensorValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YSensorValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYSensorValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## sensor→set\_highestValue() sensor→setHighestValue()sensor.set\_highestValue()

Modifie la mémoire de valeur maximale observée.

```
js function set_highestValue( newval)
node.js function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YSensor target 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.

<b>js</b>	function <b>set_logFrequency( newval)</b>
<b>node.js</b>	function <b>set_logFrequency( newval)</b>
<b>php</b>	function <b>set_logFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_logFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) setLogFrequency : (NSString*) <b>newval</b>
<b>pas</b>	function <b>set_logFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logFrequency( string newval)</b>
<b>java</b>	int <b>set_logFrequency( String newval)</b>
<b>py</b>	def <b>set_logFrequency( newval)</b>
<b>cmd</b>	<b>YSensor target set_logFrequency newval</b>

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.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YSensor 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 du senseur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_lowestValue()****YSensor****sensor→setLowestValue()sensor.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
js function set_lowestValue( newval)
nodejs function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YSensor target set_lowestValue newval
```

**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.

```
js function set_reportFrequency( newval)
nodejs function set_reportFrequency( newval)
php function set_reportFrequency( $newval)
cpp int set_reportFrequency( const string& newval)
m -(int) setReportFrequency : (NSString*) newval
pas function set_reportFrequency( newval: string): integer
vb function set_reportFrequency( ByVal newval As String) As Integer
cs int set_reportFrequency( string newval)
java int set_reportFrequency( String newval)
py def set_reportFrequency( newval)
cmd YSensor target 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

Modifie la résolution des valeurs physique mesurées.

js	function <b>set_resolution( newval)</b>
nodejs	function <b>set_resolution( newval)</b>
php	function <b>set_resolution( \$newval)</b>
cpp	int <b>set_resolution( double newval)</b>
m	-(int) <b>setResolution : (double) newval</b>
pas	function <b>set_resolution( newval: double): integer</b>
vb	function <b>set_resolution( ByVal newval As Double) As Integer</b>
cs	int <b>set_resolution( double newval)</b>
java	int <b>set_resolution( double newval)</b>
py	def <b>set_resolution( newval)</b>
cmd	YSensor <b>target set_resolution newval</b>

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).

js	function <b>set(userData)</b> ( <b>data</b> )
nodejs	function <b>set(userData)</b> ( <b>data</b> )
php	function <b>set(userData)</b> ( <b>\$data</b> )
cpp	void <b>set(userData)</b> (void* <b>data</b> )
m	-(void) <b>setUserData</b> : (void*) <b>data</b>
pas	procedure <b>set(userData)</b> ( <b>data</b> : Tobject)
vb	procedure <b>set(userData)</b> (ByVal <b>data</b> As Object)
cs	void <b>set(userData)</b> (object <b>data</b> )
java	void <b>set(userData)</b> (Object <b>data</b> )
py	def <b>set(userData)</b> ( <b>data</b> )

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()****YSensor**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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()YServo.FindServo()

**YServo**

Permet de retrouver un servo d'après un identifiant donné.

js	function <b>yFindServo( func)</b>
node.js	function <b>FindServo( func)</b>
php	function <b>yFindServo( \$func)</b>
cpp	YServo* <b>yFindServo( const string&amp; func)</b>
m	YServo* <b>yFindServo( NSString* func)</b>
pas	function <b>yFindServo( func: string): TYServo</b>
vb	function <b>yFindServo( ByVal func As String) As YServo</b>
cs	YServo <b>FindServo( string func)</b>
java	YServo <b>FindServo( String func)</b>
py	def <b>FindServo( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le servo soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le servo sans ambiguïté

### Retourne :

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

## YServo.FirstServo() yFirstServo()YServo.FirstServo()

YServo

Commence l'énumération des servo accessibles par la librairie.

```
js function yFirstServo( )
node.js function FirstServo( )
php function yFirstServo( )
cpp YServo* yFirstServo( )
m YServo* yFirstServo( )
pas function yFirstServo( ): TYServo
vb function yFirstServo( ) As YServo
cs YServo FirstServo( )
java YServo FirstServo( )
def FirstServo( )
```

Utiliser la fonction `YServo.nextServo()` pour itérer sur les autres servo.

### Retourne :

un pointeur sur un objet `YServo`, correspondant à le premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

**servo→describe()servo.describe()****YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format TYPE (NAME )=SERIAL.FUNCTIONID.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le servo      (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**servo→get\_advertisedValue()****YServo****servo→advertisedValue()servo.get\_advertisedValue()**

Retourne la valeur courante du servo (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
node.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 YServo target 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\$.

js	function <b>get_enabled( )</b>
nodejs	function <b>get_enabled( )</b>
php	function <b>get_enabled( )</b>
cpp	Y_ENABLED_enum <b>get_enabled( )</b>
m	-(Y_ENABLED_enum) enabled
pas	function <b>get_enabled( )</b> : Integer
vb	function <b>get_enabled( )</b> As Integer
cs	int <b>get_enabled( )</b>
java	int <b>get_enabled( )</b>
py	def <b>get_enabled( )</b>
cmd	YServo target <b>get_enabled</b>

**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.

```
js function get_enabledAtPowerOn( )  
nodejs function get_enabledAtPowerOn( )  
php function get_enabledAtPowerOn( )  
cpp Y_ENABLEDATPOWERON_enum get_enabledAtPowerOn( )  
m -(Y_ENABLEDATPOWERON_enum) enabledAtPowerOn  
pas function get_enabledAtPowerOn( ): Integer  
vb function get_enabledAtPowerOn( ) As Integer  
cs int get_enabledAtPowerOn( )  
java int get_enabledAtPowerOn( )  
py def get_enabledAtPowerOn( )  
cmd YServo target get_enabledAtPowerOn
```

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

## **servo→getErrorMessage()**

### **servo→errorMessage()servo.getErrorMessage()**

YServo

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
js function getErrorMessage( )  
nodejs function getErrorMessage( )  
php function getErrorMessage( )  
cpp string getErrorMessage( )  
m -(NSString*) errorMessage  
pas function getErrorMessage( ): string  
vb function getErrorMessage( ) As String  
cs string getErrorMessage( )  
java String getErrorMessage( )  
py def getErrorMessage( )
```

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.

js	function <b>get_errorType( )</b>
node.js	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	YRETCODE <b>get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	YRETCODE <b>get_errorType( )</b>
java	int <b>get_errorType( )</b>
py	def <b>get_errorType( )</b>

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.

```
js function get_friendlyName( )  
nodejs function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

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.

js	function get_functionDescriptor( )
nodejs	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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.

```
js function get_hardwareId( )  
node.js function get_hardwareId( )  
php function get_hardwareId( )  
cpp string get_hardwareId( )  
m -(NSString*) hardwareId  
vb function get_hardwareId( ) As String  
cs string get_hardwareId( )  
java String get_hardwareId( )  
py def get_hardwareId( )
```

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()**  
**servo→logicalName()servo.get\_logicalName()****YServo**

Retourne le nom logique du servo.

```
js function get_logicalName( )
nodejs function get_logicalName( )
php function get_logicalName( )
cpp string get_logicalName( )
m -(NSString*) logicalName
pas function get_logicalName( ): string
vb function get_logicalName( ) As String
cs string get_logicalName( )
java String get_logicalName( )
py def get_logicalName( )
cmd YServo target 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()**  
**servo→module()servo.get\_module()****YServo**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**servo→get\_module\_async()****YServo****servo→module\_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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()**  
**servo→neutral()servo.get\_neutral()****YServo**

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

js	function <b>get_neutral( )</b>
node.js	function <b>get_neutral( )</b>
php	function <b>get_neutral( )</b>
cpp	int <b>get_neutral( )</b>
m	-(int) neutral
pas	function <b>get_neutral( ): LongInt</b>
vb	function <b>get_neutral( ) As Integer</b>
cs	int <b>get_neutral( )</b>
java	int <b>get_neutral( )</b>
py	def <b>get_neutral( )</b>
cmd	<b>YServo target get_neutral</b>

**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()**

### **servo→position()servo.get\_position()**

YServo

Retourne la position courante du servo.

```
js function get_position( )
nodejs function get_position( )
php function get_position( )
cpp int get_position( )
m -(int) position
pas function get_position( ): LongInt
vb function get_position( ) As Integer
cs int get_position( )
java int get_position( )
py def get_position( )
cmd YServo target get_position
```

**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.

```
js function get_positionAtPowerOn( )
nodejs function get_positionAtPowerOn( )
php function get_positionAtPowerOn( )
cpp int get_positionAtPowerOn( )
m -(int) positionAtPowerOn
pas function get_positionAtPowerOn( ): LongInt
vb function get_positionAtPowerOn( ) As Integer
cs int get_positionAtPowerOn( )
java int get_positionAtPowerOn( )
py def get_positionAtPowerOn( )
cmd YServo target 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()**  
**servo→range()servo.get\_range()****YServo**

Retourne la plage d'utilisation du servo.

js	function <b>get_range( )</b>
nodejs	function <b>get_range( )</b>
php	function <b>get_range( )</b>
cpp	int <b>get_range( )</b>
m	-(int) range
pas	function <b>get_range( )</b> : LongInt
vb	function <b>get_range( )</b> As Integer
cs	int <b>get_range( )</b>
java	int <b>get_range( )</b>
py	def <b>get_range( )</b>
cmd	YServo target <b>get_range</b>

**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)**  
**servo→userData()servo.get(userData)****YServo**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData) {  
m -(void*) userData  
pas function get(userData): Tobject  
vb function get(userData) As Object  
cs object get(userData)  
java Object get(userData)  
py def get(userData)
```

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.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()****YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()

YServo

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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.

**servo→move()servo.move()****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

<b>js</b>	function <b>move( target, ms_duration)</b>
<b>node.js</b>	function <b>move( target, ms_duration)</b>
<b>php</b>	function <b>move( \$target, \$ms_duration)</b>
<b>cpp</b>	int <b>move( int target, int ms_duration)</b>
<b>m</b>	- <b>(int) move : (int) target : (int) ms_duration</b>
<b>pas</b>	function <b>move( target: LongInt, ms_duration: LongInt): integer</b>
<b>vb</b>	function <b>move( ByVal target As Integer,</b> <b>                  ByVal ms_duration As Integer) As Integer</b>
<b>cs</b>	int <b>move( int target, int ms_duration)</b>
<b>java</b>	int <b>move( int target, int ms_duration)</b>
<b>py</b>	def <b>move( target, ms_duration)</b>
<b>cmd</b>	<b>YServo target move target ms_duration</b>

**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()`.

js	function <b>nextServo</b> ( )
nodejs	function <b>nextServo</b> ( )
php	function <b>nextServo</b> ( )
cpp	YServo * <b>nextServo</b> ( )
m	-(YServo*) <b>nextServo</b>
pas	function <b>nextServo</b> ( ): TYServo
vb	function <b>nextServo</b> ( ) As YServo
cs	YServo <b>nextServo</b> ( )
java	YServo <b>nextServo</b> ( )
py	def <b>nextServo</b> ( )

**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.

<code>js</code>	<b>function registerValueCallback( <b>callback</b>)</b>
<code>node.js</code>	<b>function registerValueCallback( <b>callback</b>)</b>
<code>php</code>	<b>function registerValueCallback( <b>\$callback</b>)</b>
<code>cpp</code>	<b>int registerValueCallback( YServoValueCallback <b>callback</b>)</b>
<code>m</code>	<b>-(int) registerValueCallback : (YServoValueCallback) <b>callback</b></b>
<code>pas</code>	<b>function registerValueCallback( <b>callback</b>: TYServoValueCallback): LongInt</b>
<code>vb</code>	<b>function registerValueCallback( ) As Integer</b>
<code>cs</code>	<b>int registerValueCallback( ValueCallback <b>callback</b>)</b>
<code>java</code>	<b>int registerValueCallback( UpdateCallback <b>callback</b>)</b>
<code>py</code>	<b>def registerValueCallback( <b>callback</b>)</b>

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

#### **Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**servo→set\_enabled()  
servo→setEnabled()servo.set\_enabled()****YServo**

Démarrer ou arrête le \$FUNCTION\$.

js	function <b>set_enabled( newval)</b>
node.js	function <b>set_enabled( newval)</b>
php	function <b>set_enabled( \$newval)</b>
cpp	int <b>set_enabled( Y_ENABLED_enum newval)</b>
m	-(int) setEnabled : (Y_ENABLED_enum) <b>newval</b>
pas	function <b>set_enabled( newval: Integer): integer</b>
vb	function <b>set_enabled( ByVal newval As Integer) As Integer</b>
cs	int <b>set_enabled( int newval)</b>
java	int <b>set_enabled( int newval)</b>
py	def <b>set_enabled( newval)</b>
cmd	<b>YServo target set_enabled newval</b>

**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.

js	function <b>set_enabledAtPowerOn( newval)</b>
node.js	function <b>set_enabledAtPowerOn( newval)</b>
php	function <b>set_enabledAtPowerOn( \$newval)</b>
cpp	int <b>set_enabledAtPowerOn( Y_ENABLEDATPOWERON_enum newval)</b>
m	-(int) <b>setEnabledAtPowerOn : (Y_ENABLEDATPOWERON_enum) newval</b>
pas	function <b>set_enabledAtPowerOn( newval: Integer): integer</b>
vb	function <b>set_enabledAtPowerOn( ByVal newval As Integer) As Integer</b>
cs	int <b>set_enabledAtPowerOn( int newval)</b>
java	int <b>set_enabledAtPowerOn( int newval)</b>
py	def <b>set_enabledAtPowerOn( newval)</b>
cmd	YServo <b>target set_enabledAtPowerOn newval</b>

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_logicalName()****servo→setLogicalName()servo.set\_logicalName()****YServo**

Modifie le nom logique du servo.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YServo 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 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

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

js	function <b>set_neutral( newval)</b>
nodejs	function <b>set_neutral( newval)</b>
php	function <b>set_neutral( \$newval)</b>
cpp	int <b>set_neutral( int newval)</b>
m	-(int) setNeutral : (int) <b>newval</b>
pas	function <b>set_neutral( newval: LongInt): integer</b>
vb	function <b>set_neutral( ByVal newval As Integer) As Integer</b>
cs	int <b>set_neutral( int newval)</b>
java	int <b>set_neutral( int newval)</b>
py	def <b>set_neutral( newval)</b>
cmd	YServo target <b>set_neutral newval</b>

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.

js	function <b>set_position( newval)</b>
node.js	function <b>set_position( newval)</b>
php	function <b>set_position( \$newval)</b>
cpp	int <b>set_position( int newval)</b>
m	-(int) setPosition : (int) <b>newval</b>
pas	function <b>set_position( newval: LongInt): integer</b>
vb	function <b>set_position( ByVal newval As Integer) As Integer</b>
cs	int <b>set_position( int newval)</b>
java	int <b>set_position( int newval)</b>
py	def <b>set_position( newval)</b>
cmd	<b>YServo target set_position newval</b>

**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.

js	function <b>set_positionAtPowerOn( newval)</b>
node.js	function <b>setPositionAtPowerOn( newval)</b>
php	function <b>set_positionAtPowerOn( \$newval)</b>
cpp	int <b>set_positionAtPowerOn( int newval)</b>
m	-(int) setPositionAtPowerOn : (int) <b>newval</b>
pas	function <b>set_positionAtPowerOn( newval: LongInt): integer</b>
vb	function <b>set_positionAtPowerOn( ByVal newval As Integer) As Integer</b>
cs	int <b>set_positionAtPowerOn( int newval)</b>
java	int <b>set_positionAtPowerOn( int newval)</b>
py	def <b>set_positionAtPowerOn( newval)</b>
cmd	YServo <b>target set_positionAtPowerOn newval</b>

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.

js	function <b>set_range( newval)</b>
node.js	function <b>set_range( newval)</b>
php	function <b>set_range( \$newval)</b>
cpp	int <b>set_range( int newval)</b>
m	-(int) setRange : (int) newval
pas	function <b>set_range( newval: LongInt): integer</b>
vb	function <b>set_range( ByVal newval As Integer) As Integer</b>
cs	int <b>set_range( int newval)</b>
java	int <b>set_range( int newval)</b>
py	def <b>set_range( newval)</b>
cmd	<b>YServo target set_range newval</b>

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).

js	function set(userData( data)
node.js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

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()****YServo**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()****YTemperature****yFindTemperature()YTemperature.FindTemperature()**

Permet de retrouver un capteur de température d'après un identifiant donné.

<b>js</b>	function <b>yFindTemperature( func)</b>
<b>node.js</b>	function <b>FindTemperature( func)</b>
<b>php</b>	function <b>yFindTemperature( \$func)</b>
<b>cpp</b>	YTemperature* <b>yFindTemperature( const string&amp; func)</b>
<b>m</b>	YTemperature* <b>yFindTemperature( NSString* func)</b>
<b>pas</b>	function <b>yFindTemperature( func: string): TYTemperature</b>
<b>vb</b>	function <b>yFindTemperature( ByVal func As String) As YTemperature</b>
<b>cs</b>	YTemperature <b>FindTemperature( string func)</b>
<b>java</b>	YTemperature <b>FindTemperature( String func)</b>
<b>py</b>	def <b>FindTemperature( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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()

### yFirstTemperature()YTemperature.FirstTemperature()

## YTemperature

Commence l'énumération des capteurs de température accessibles par la librairie.

js	function <b>yFirstTemperature( )</b>
node.js	function <b>FirstTemperature( )</b>
php	function <b>yFirstTemperature( )</b>
cpp	YTemperature* <b>yFirstTemperature( )</b>
m	YTemperature* <b>yFirstTemperature( )</b>
pas	function <b>yFirstTemperature( ): YTemperature</b>
vb	function <b>yFirstTemperature( ) As YTemperature</b>
cs	YTemperature <b>FirstTemperature( )</b>
java	YTemperature <b>FirstTemperature( )</b>
py	def <b>FirstTemperature( )</b>

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

#### Retourne :

un pointeur sur un objet `YTemperature`, correspondant à le premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

## temperature→calibrateFromPoints() 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php   function calibrateFromPoints( $rawValues, $refValues)
cpp    int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)
m     -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb    procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java  int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py    def calibrateFromPoints( rawValues, refValues)
cmd   YTemperature target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
node.js	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(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 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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YTemperature target 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()**  
**temperature→currentRawValue()**  
**temperature.get\_currentRawValue()**

**YTemperature**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

js	function <b>get_currentRawValue( )</b>
node.js	function <b>get_currentRawValue( )</b>
php	function <b>get_currentRawValue( )</b>
cpp	double <b>get_currentRawValue( )</b>
m	-(double) currentRawValue
pas	function <b>get_currentRawValue( ): double</b>
vb	function <b>get_currentRawValue( ) As Double</b>
cs	double <b>get_currentRawValue( )</b>
java	double <b>get_currentRawValue( )</b>
py	def <b>get_currentRawValue( )</b>
cmd	YTemperature target <b>get_currentRawValue</b>

**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.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YTemperature target 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.

```
js function getErrorMessage( )
nodejs function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ): string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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.

```
js function get_errorType( )
nodejs function get_errorType( )
php function get_errorType( )
cpp YRETCODE get_errorType( )
pas function get_errorType( ): YRETCODE
vb function get_errorType( ) As YRETCODE
cs YRETCODE get_errorType( )
java int get_errorType( )
py def get_errorType( )
```

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.

```
js function get_friendlyName( )
node.js function get_friendlyName( )
php function get_friendlyName( )
cpp string get_friendlyName( )
m -(NSString*) friendlyName
cs string get_friendlyName( )
java String get_friendlyName( )
py def get_friendlyName( )
```

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId()</b> {
node.js	function <b>get_functionId()</b> {
php	function <b>get_functionId()</b> {
cpp	string <b>get_functionId()</b> {
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b> {
java	String <b>get_functionId()</b> {
py	def <b>get_functionId()</b> {

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.

js	function get_hardwareId( )
node.js	function get_hardwareId( )
php	function get_hardwareId( )
cpp	string get_hardwareId( )
m	-(NSString*) hardwareId
vb	function get_hardwareId( ) As String
cs	string get_hardwareId( )
java	String get_hardwareId( )
py	def get_hardwareId( )

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.

js	function <b>get_highestValue( )</b>
node.js	function <b>get_highestValue( )</b>
php	function <b>get_highestValue( )</b>
cpp	double <b>get_highestValue( )</b>
m	-(double) highestValue
pas	function <b>get_highestValue( )</b> : double
vb	function <b>get_highestValue( )</b> As Double
cs	double <b>get_highestValue( )</b>
java	double <b>get_highestValue( )</b>
py	def <b>get_highestValue( )</b>
cmd	YTemperature target <b>get_highestValue</b>

**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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YTemperature target 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.

```
js   function get_logicalName( )  
node.js function get_logicalName( )  
php  function get_logicalName( )  
cpp   string get_logicalName( )  
m    -(NSString*) logicalName  
pas   function get_logicalName( ): string  
vb    function get_logicalName( ) As String  
cs    string get_logicalName( )  
java  String get_logicalName( )  
py    def get_logicalName( )  
cmd   YTemperature target 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.

```
js function get_lowestValue( )  
nodejs function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YTemperature target 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.

js	function <b>get_module( )</b>
nodejs	function <b>get_module( )</b>
php	function <b>get_module( )</b>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module( ): TYModule</b>
vb	function <b>get_module( ) As YModule</b>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**temperature→get\_module\_async()**  
**temperature→module\_async()****YTemperature**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YTemperature target get_recordedData startTime endTime

```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

#### Paramètres :

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

#### Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**temperature→get\_reportFrequency()**  
**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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YTemperature target get_reportFrequency
```

**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.

```
js   function get_resolution( )  
node.js function get_resolution( )  
php  function get_resolution( )  
cpp   double get_resolution( )  
m    -(double) resolution  
pas   function get_resolution( ): double  
vb    function get_resolution( ) As Double  
cs    double get_resolution( )  
java  double get_resolution( )  
py    def get_resolution( )  
cmd   YTemperature target 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

```
js function get_sensorType( )  
nodejs function get_sensorType( )  
php function get_sensorType( )  
cpp Y_SENSORTYPE_enum get_sensorType( )  
m -(Y_SENSORTYPE_enum) sensorType  
pas function get_sensorType( ): Integer  
vb function get_sensorType( ) As Integer  
cs int get_sensorType( )  
java int get_sensorType( )  
py def get_sensorType( )  
cmd YTemperature target 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.

```
js function get_unit( )
nodejs function get_unit( )
php function get_unit( )
cpp string get_unit( )
m -(NSString*) unit
pas function get_unit( ): string
vb function get_unit( ) As String
cs string get_unit( )
java String get_unit( )
py def get_unit( )
cmd YTemperature target 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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData);  
m -(void*) userData;  
pas function get(userData): Tobject;  
vb function get(userData) As Object;  
cs object get(userData);  
java Object get(userData);  
py def get(userData):
```

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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	- <b>(BOOL) isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	<b>def isOnline( )</b>

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()****YTemperature**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

js	function load( msValidity)
nodejs	function load( msValidity)
php	function load( \$msValidity)
cpp	YRETCODE load( int msValidity)
m	-(YRETCODE) load : (int) msValidity
pas	function load( msValidity: integer): YRETCODE
vb	function load( ByVal msValidity As Integer) As YRETCODE
cs	YRETCODE load( int msValidity)
java	int load( long msValidity)
py	def load( msValidity)

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function loadCalibrationPoints( var rawValues: TDoubleArray,
                                      var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( )
cs    int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
py    def loadCalibrationPoints( rawValues, refValues)
cmd   YTemperature target loadCalibrationPoints rawValues refValues

```

### 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()

## YTemperature

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextTemperature()</b>
nodejs	function <b>nextTemperature()</b>
php	function <b>nextTemperature()</b>
cpp	YTemperature * <b>nextTemperature()</b>
m	- <b>(YTemperature*) nextTemperature</b>
pas	function <b>nextTemperature()</b> : TYTemperature
vb	function <b>nextTemperature()</b> As YTemperature
cs	YTemperature <b>nextTemperature()</b>
java	YTemperature <b>nextTemperature()</b>
py	<b>def nextTemperature()</b>

**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.

js	function registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YTemperatureTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YTemperatureTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYTemperatureTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <b>callback</b> )

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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YTemperatureValueCallback callback)
m -(int) registerValueCallback : (YTemperatureValueCallback) callback
pas function registerValueCallback( callback: TYTemperatureValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

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.

<b>js</b>	function <b>set_highestValue( newval)</b>
<b>node.js</b>	function <b>set_highestValue( newval)</b>
<b>php</b>	function <b>set_highestValue( \$newval)</b>
<b>cpp</b>	int <b>set_highestValue( double newval)</b>
<b>m</b>	-(int) setHighestValue : (double) <b>newval</b>
<b>pas</b>	function <b>set_highestValue( newval: double): integer</b>
<b>vb</b>	function <b>set_highestValue( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_highestValue( double newval)</b>
<b>java</b>	int <b>set_highestValue( double newval)</b>
<b>py</b>	def <b>set_highestValue( newval)</b>
<b>cmd</b>	<b>YTemperature target set_highestValue newval</b>

#### 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.

```
js function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YTemperature target 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.

<b>js</b>	function <b>set_logicalName( newval)</b>
<b>node.js</b>	function <b>set_logicalName( newval)</b>
<b>php</b>	function <b>set_logicalName( \$newval)</b>
<b>cpp</b>	int <b>set_logicalName( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setLogicalName : (NSString*) newval</b>
<b>pas</b>	function <b>set_logicalName( newval: string): integer</b>
<b>vb</b>	function <b>set_logicalName( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_logicalName( string newval)</b>
<b>java</b>	int <b>set_logicalName( String newval)</b>
<b>py</b>	def <b>set_logicalName( newval)</b>
<b>cmd</b>	<b>YTemperature target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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.

```
js function set_lowestValue( newval)
nodejs function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YTemperature target 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.

<b>js</b>	function <b>set_reportFrequency( newval)</b>
<b>node.js</b>	function <b>set_reportFrequency( newval)</b>
<b>php</b>	function <b>set_reportFrequency( \$newval)</b>
<b>cpp</b>	int <b>set_reportFrequency( const string&amp; newval)</b>
<b>m</b>	-(int) <b>setReportFrequency : (NSString*) newval</b>
<b>pas</b>	function <b>set_reportFrequency( newval: string): integer</b>
<b>vb</b>	function <b>set_reportFrequency( ByVal newval As String) As Integer</b>
<b>cs</b>	int <b>set_reportFrequency( string newval)</b>
<b>java</b>	int <b>set_reportFrequency( String newval)</b>
<b>py</b>	def <b>set_reportFrequency( newval)</b>
<b>cmd</b>	<b>YTemperature target set_reportFrequency newval</b>

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.

```
js function set_resolution( newval)
nodejs function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YTemperature target 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.

js	function <b>set_sensorType( newval)</b>
node.js	function <b>set_sensorType( newval)</b>
php	function <b>set_sensorType( \$newval)</b>
cpp	int <b>set_sensorType( Y_SENSORTYPE_enum newval)</b>
m	-(int) <b>setSensorType : (Y_SENSORTYPE_enum) newval</b>
pas	function <b>set_sensorType( newval: Integer): integer</b>
vb	function <b>set_sensorType( ByVal newval As Integer) As Integer</b>
cs	int <b>set_sensorType( int newval)</b>
java	int <b>set_sensorType( int newval)</b>
py	def <b>set_sensorType( newval)</b>
cmd	<b>YTemperature target set_sensorType newval</b>

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).

```
js function set(userData) {  
nodejs function setUserData( data)  
php function setUserData( $data)  
cpp void setUserData( void* data)  
m -(void) setUserData : (void*) data  
pas procedure set(userData: Tobject)  
vb procedure setUserData( ByVal data As Object)  
cs void set(userData: object data)  
java void setUserData( Object data)  
py 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

## 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.

```
js function wait_async( callback, context)
nodejs 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()YTilt.FindTilt()**

Permet de retrouver un inclinomètre d'après un identifiant donné.

js	function <b>yFindTilt( func)</b>
node.js	function <b>FindTilt( func)</b>
php	function <b>yFindTilt( \$func)</b>
cpp	YTilt* <b>yFindTilt( const string&amp; func)</b>
m	YTilt* <b>yFindTilt( NSString* func)</b>
pas	function <b>yFindTilt( func: string): TYTilt</b>
vb	function <b>yFindTilt( ByVal func As String) As YTilt</b>
cs	YTilt <b>FindTilt( string func)</b>
java	YTilt <b>FindTilt( String func)</b>
py	def <b>FindTilt( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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()YTilt.FirstTilt()

YTilt

Commence l'énumération des inclinomètres accessibles par la librairie.

```
js function yFirstTilt( )
node.js function FirstTilt( )
php function yFirstTilt( )
cpp YTilt* yFirstTilt( )
m YTilt* yFirstTilt( )
pas function yFirstTilt( ): TYTilt
vb function yFirstTilt( ) As YTilt
cs YTilt FirstTilt( )
java YTilt FirstTilt( )
def FirstTilt( )
```

Utiliser la fonction `YTilt.nextTilt()` pour itérer sur les autres inclinomètres.

### Retourne :

un pointeur sur un objet `YTilt`, correspondant à le 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.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                           vector<double> refValues)
m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)
py def calibrateFromPoints( rawValues, refValues)
cmd YTilt target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe()</b>
nodejs	function <b>describe()</b>
php	function <b>describe()</b>
cpp	string <b>describe()</b>
m	-(NSString*) <b>describe</b>
pas	function <b>describe()</b> : string
vb	function <b>describe()</b> As String
cs	string <b>describe()</b>
java	String <b>describe()</b>
py	def <b>describe()</b>

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant l'inclinomètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**tilt→get\_advertisedValue()**

YTilt

**tilt→advertisedValue()tilt.get\_advertisedValue()**

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YTilt target get_advertisedValue
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**tilt→get\_currentRawValue()** YTilt  
**tilt→currentRawValue()tilt.get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
js function get_currentRawValue( )  
node.js function get_currentRawValue( )  
php function get_currentRawValue( )  
cpp double get_currentRawValue( )  
m -(double) currentRawValue  
pas function get_currentRawValue( ): double  
vb function get_currentRawValue( ) As Double  
cs double get_currentRawValue( )  
java double get_currentRawValue( )  
py def get_currentRawValue( )  
cmd YTilt target get_currentRawValue
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**tilt→get\_currentValue()**

YTilt

**tilt→currentValue()tilt.get\_currentValue()**

Retourne la valeur actuelle de l'inclinaison.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YTilt target get_currentValue
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'inclinaison

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**tilt→getErrorMessage()** **YTilt**  
**tilt→errorMessage()tilt.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
js function getErrorMessage( )  
nodejs function getErrorMessage( )  
php function getErrorMessage( )  
cpp string getErrorMessage( )  
m -(NSString*) errorMessage  
pas function getErrorMessage( ):string  
vb function getErrorMessage( ) As String  
cs string getErrorMessage( )  
java String getErrorMessage( )  
py def getErrorMessage( )
```

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() tilt→errorType()tilt.get\_errorType()

YTilt

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ):YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

**tilt→get\_friendlyName()  
tilt→friendlyName()tilt.get\_friendlyName()****YTilt**

Retourne un identifiant global de l'inclinomètre au format NOM\_MODULE . NOM\_FONCTION.

js	function get_friendlyName( )
node.js	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

Le chaîne renvoyée utilise soit les noms logiques du module et de l'inclinomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'inclinomètre (par exemple: MyCustomName.relay1)

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FRIENDLYNAME\_INVALID.

## tilt→get\_functionDescriptor() tilt→functionDescriptor()tilt.get\_functionDescriptor()

YTilt

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
js function get_functionDescriptor( )
nodejs function get_functionDescriptor( )
php function get_functionDescriptor( )
cpp YFUN_DESCR get_functionDescriptor( )
m -(YFUN_DESCR) functionDescriptor
pas function get_functionDescriptor( ): YFUN_DESCR
vb function get_functionDescriptor( ) As YFUN_DESCR
cs YFUN_DESCR get_functionDescriptor( )
java String get_functionDescriptor( )
py def get_functionDescriptor( )
```

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

**tilt→get\_functionId()  
tilt→functionId()tilt.get\_functionId()**

YTilt

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**tilt→get\_hardwareId()**

YTilt

**tilt→hardwareId()tilt.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'inclinomètre au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId()</b>
nodejs	function <b>get_hardwareId()</b>
php	function <b>get_hardwareId()</b>
cpp	string <b>get_hardwareId()</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId()</b> As String
cs	string <b>get_hardwareId()</b>
java	String <b>get_hardwareId()</b>
py	def <b>get_hardwareId()</b>

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'inclinomètre (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**tilt→get\_highestValue()**  
**tilt→highestValue()tilt.get\_highestValue()**

YTilt

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

```
js function get_highestValue( )  
node.js function get_highestValue( )  
php function get_highestValue( )  
cpp double get_highestValue( )  
m -(double) highestValue  
pas function get_highestValue( ): double  
vb function get_highestValue( ) As Double  
cs double get_highestValue( )  
java double get_highestValue( )  
py def get_highestValue( )  
cmd YTilt target 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.

```
js function get_logFrequency( )  
nodejs function get_logFrequency( )  
php function get_logFrequency( )  
cpp string get_logFrequency( )  
m -(NSString*) logFrequency  
pas function get_logFrequency( ): string  
vb function get_logFrequency( ) As String  
cs string get_logFrequency( )  
java String get_logFrequency( )  
py def get_logFrequency( )  
cmd YTilt target get_logFrequency
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**tilt→get\_logicalName()**  
**tilt→logicalName()tilt.get\_logicalName()**

YTilt

Retourne le nom logique de l'inclinomètre.

```
js function get_logicalName( )  
node.js function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YTilt target 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() tilt→lowestValue()tilt.get\_lowestValue()

YTilt

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

```
js function get_lowestValue( )  
nodejs function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YTilt target get_lowestValue
```

### Retourne :

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**tilt→get\_module()  
tilt→module()tilt.get\_module()**

YTilt

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module( )
node.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

## tilt→get\_module\_async() tilt→module\_async()

YTilt

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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.

## tilt→get\_recordedData() YTilt tilt→recordedData() tilt.get\_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php   function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m     -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime
pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java YDataSet get_recordedData( long startTime, long endTime)
py   def get_recordedData( startTime, endTime)
cmd  YTilt target 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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YTilt target get_reportFrequency
```

**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()  
tilt→resolution()tilt.get\_resolution()****YTilt**

Retourne la résolution des valeurs mesurées.

```
js function get_resolution( )
node.js function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YTilt target 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.

```
js function get_unit( )
nodejs function get_unit( )
php function get_unit( )
cpp string get_unit( )
m -(NSString*) unit
pas function get_unit( ): string
vb function get_unit( ) As String
cs string get_unit( )
java String get_unit( )
py def get_unit( )
cmd YTilt target 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) tilt→userData()tilt.get(userData)

YTilt

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
js function get(userData) 
nodejs function get(userData) 
php function get(userData) 
cpp void * get(userData) 
m -(void*) userData 
pas function get(userData): Tobject 
vb function get(userData) As Object 
cs object get(userData) 
java Object get(userData) 
py def get(userData)
```

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.

```
js function isOnline( )
node.js function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

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()

YTilt

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function loadCalibrationPoints( var rawValues: TDoubleArray,
                                      var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( )
cs    int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
py    def loadCalibrationPoints( rawValues, refValues)
cmd   YTilt target loadCalibrationPoints rawValues refValues

```

### 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()

## YTilt

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextTilt( )</b>
nodejs	function <b>nextTilt( )</b>
php	function <b>nextTilt( )</b>
cpp	YTilt * <b>nextTilt( )</b>
m	-YTilt* <b>nextTilt</b>
pas	function <b>nextTilt( )</b> : TYTilt
vb	function <b>nextTilt( )</b> As YTilt
cs	YTilt <b>nextTilt( )</b>
java	YTilt <b>nextTilt( )</b>
py	def <b>nextTilt( )</b>

**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.

```

js   function registerTimedReportCallback( callback )
node.js function registerTimedReportCallback( callback )
php  function registerTimedReportCallback( $callback )
cpp   int registerTimedReportCallback( YTiltTimedReportCallback callback )
m     -(int) registerTimedReportCallback : (YTiltTimedReportCallback) callback
pas   function registerTimedReportCallback( callback: TYTiltTimedReportCallback): LongInt
vb    function registerTimedReportCallback( ) As Integer
cs    int registerTimedReportCallback( TimedReportCallback callback )
java  int registerTimedReportCallback( TimedReportCallback callback )
py    def registerTimedReportCallback( callback )

```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

## tilt→registerValueCallback() tilt.registerValueCallback()

YTilt

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YTiltValueCallback callback)
m -(int) registerValueCallback : (YTiltValueCallback) callback
pas function registerValueCallback( callback: TYTiltValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

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.

```
js function set_highestValue( newval)
nodejs function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YTilt target set_highestValue newval
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## tilt→set\_logFrequency() tilt→setLogFrequency()tilt.set\_logFrequency()

YTilt

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
js function set_logFrequency( newval)
node.js function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YTilt target set_logFrequency newval
```

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() tilt→setLogicalName()tilt.set\_logicalName()

YTilt

Modifie le nom logique de l'inclinomètre.

js	<code>function set_logicalName( newval)</code>
node.js	<code>function set_logicalName( newval)</code>
php	<code>function set_logicalName( \$newval)</code>
cpp	<code>int set_logicalName( const string&amp; newval)</code>
m	<code>-(int) setLogicalName : (NSString*) newval</code>
pas	<code>function set_logicalName( newval: string): integer</code>
vb	<code>function set_logicalName( ByVal newval As String) As Integer</code>
cs	<code>int set_logicalName( string newval)</code>
java	<code>int set_logicalName( String newval)</code>
py	<code>def set_logicalName( newval)</code>
cmd	<code>YTilt target set_logicalName newval</code>

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()  
tilt→setLowestValue()tilt.set\_lowestValue()**

YTilt

Modifie la mémoire de valeur minimale observée.

```
js function set_lowestValue( newval)
node.js function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YTilt target 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.

<b>js</b>	<code>function set_reportFrequency( newval)</code>
<b>node.js</b>	<code>function set_reportFrequency( newval)</code>
<b>php</b>	<code>function set_reportFrequency( \$newval)</code>
<b>cpp</b>	<code>int set_reportFrequency( const string&amp; newval)</code>
<b>m</b>	<code>-(int) setReportFrequency : (NSString*) newval</code>
<b>pas</b>	<code>function set_reportFrequency( newval: string): integer</code>
<b>vb</b>	<code>function set_reportFrequency( ByVal newval As String) As Integer</code>
<b>cs</b>	<code>int set_reportFrequency( string newval)</code>
<b>java</b>	<code>int set_reportFrequency( String newval)</code>
<b>py</b>	<code>def set_reportFrequency( newval)</code>
<b>cmd</b>	<code>YTilt target set_reportFrequency newval</code>

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() tilt→setResolution()tilt.set\_resolution()

YTilt

Modifie la résolution des valeurs physique mesurées.

```
js function set_resolution( newval)
node.js function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YTilt target 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).

```
js function setUserData( data)
nodejs function setUserData( data)
php function setUserData( $data)
cpp void setUserData( void* data)
m -(void) setUserData : (void*) data
pas procedure setUserData( data: Tobject)
vb procedure setUserData( ByVal data As Object)
cs void setUserData( object data)
java void setUserData( Object data)
py def setUserData( data)
```

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()****YTilt**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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() YVoc.FindVoc()

YVoc

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

js	<code>function yFindVoc( func)</code>
node.js	<code>function FindVoc( func)</code>
php	<code>function yFindVoc( \$func)</code>
cpp	<code>YVoc* yFindVoc( const string&amp; func)</code>
m	<code>YVoc* yFindVoc( NSString* func)</code>
pas	<code>function yFindVoc( func: string): TYVoc</code>
vb	<code>function yFindVoc( ByVal func As String) As YVoc</code>
cs	<code>YVoc FindVoc( string func)</code>
java	<code>YVoc FindVoc( String func)</code>
py	<code>def FindVoc( 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 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() YVoc.FirstVoc()**

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

```
js function yFirstVoc( )
nodejs function FirstVoc( )
php function yFirstVoc( )
cpp YVoc* yFirstVoc( )
m YVoc* yFirstVoc( )
pas function yFirstVoc( ): TYVoc
vb function yFirstVoc( ) As YVoc
cs YVoc FirstVoc( )
java YVoc FirstVoc( )
py def FirstVoc( )
```

Utiliser la fonction `YVoc.nextVoc()` pour itérer sur les autres capteurs de Composés Organiques Volatils.

**Retourne :**

un pointeur sur un objet `YVOC`, correspondant à le 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.

```

js function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
cpp int calibrateFromPoints( vector<double> rawValues,
                           vector<double> refValues)

m -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py def calibrateFromPoints( rawValues, refValues)
cmd YVoc target 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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)=RELAY01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex:  
Relay(MyCustomName.relay1)=RELAY01-123456.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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YVoc target 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).

```
js function get_currentRawValue( )  
nodejs function get_currentRawValue( )  
php function get_currentRawValue( )  
cpp double get_currentRawValue( )  
m -(double) currentRawValue  
pas function get_currentRawValue( ): double  
vb function get_currentRawValue( ) As Double  
cs double get_currentRawValue( )  
java double get_currentRawValue( )  
py def get_currentRawValue( )  
cmd YVoc target get_currentRawValue
```

**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()  
voc→currentValue()voc.get\_currentValue()****YVoc**

Retourne la mesure actuelle du taux de VOC estimé.

```
js function get_currentValue( )
node.js function get_currentValue( )
php function get_currentValue( )
cpp double get_currentValue( )
m -(double) currentValue
pas function get_currentValue( ): double
vb function get_currentValue( ) As Double
cs double get_currentValue( )
java double get_currentValue( )
py def get_currentValue( )
cmd YVoc target 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.

```
js function get_errorMessage( )  
nodejs function get_errorMessage( )  
php function get_errorMessage( )  
cpp string get_errorMessage( )  
m -(NSString*) errorMessage  
pas function get_errorMessage( ): string  
vb function get_errorMessage( ) As String  
cs string get_errorMessage( )  
java String get_errorMessage( )  
py def get_errorMessage( )
```

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 Organiques Volatils.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

**voc→get\_friendlyName()****YVoc****voc→friendlyName()voc.get\_friendlyName()**

Retourne un identifiant global du capteur de Composés Organiques Volatils au format NOM\_MODULE.NOM\_FONCTION.

js	function <b>get_friendlyName( )</b>
nodejs	function <b>get_friendlyName( )</b>
php	function <b>get_friendlyName( )</b>
cpp	string <b>get_friendlyName( )</b>
m	-(NSString*) friendlyName
cs	string <b>get_friendlyName( )</b>
java	String <b>get_friendlyName( )</b>
py	def <b>get_friendlyName( )</b>

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.

js	function get_functionDescriptor( )
nodejs	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

Par example `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()  
voc→hardwareId()voc.get\_hardwareId()****YVoc**

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.

```
js function get_hardwareId( )  
nodejs function get_hardwareId( )  
php function get_hardwareId( )  
cpp string get_hardwareId( )  
m -(NSString*) hardwareId  
vb function get_hardwareId( ) As String  
cs string get_hardwareId( )  
java String get_hardwareId( )  
py def get_hardwareId( )
```

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.

```
js function get_highestValue( )
nodejs function get_highestValue( )
php function get_highestValue( )
cpp double get_highestValue( )
m -(double) highestValue
pas function get_highestValue( ): double
vb function get_highestValue( ) As Double
cs double get_highestValue( )
java double get_highestValue( )
py def get_highestValue( )
cmd YVoc target 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()**  
**voc→logFrequency()voc.get\_logFrequency()****YVoc**

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.

```
js function get_logFrequency( )  
nodejs function get_logFrequency( )  
php function get_logFrequency( )  
cpp string get_logFrequency( )  
m -(NSString*) logFrequency  
pas function get_logFrequency( ):string  
vb function get_logFrequency( ) As String  
cs string get_logFrequency( )  
java String get_logFrequency( )  
py def get_logFrequency( )  
cmd YVoc target 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.

```
js function get_logicalName( )  
nodejs function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YVoc target 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()  
voc→lowestValue()voc.get\_lowestValue()****YVoc**

Retourne la valeur minimale observée pour le taux de VOC estimé.

```
js function get_lowestValue( )  
node.js function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YVoc target 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()**  
**voc→module()voc.get\_module()****YVoc**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	<code>function get_module( )</code>
nodejs	<code>function get_module( )</code>
php	<code>function get_module( )</code>
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	<code>function get_module( ): TYModule</code>
vb	<code>function get_module( ) As YModule</code>
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**voc→get\_module\_async()**  
**voc→module\_async()****YVoc**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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.

```

js   function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php  function get_recordedData( $startTime, $endTime)
cpp   YDataSet get_recordedData( s64 startTime, s64 endTime)
m    -(YDataSet*) recordedData : (s64) startTime
                  : (s64) endTime

pas  function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb   function get_recordedData( ) As YDataSet
cs   YDataSet get_recordedData( long startTime, long endTime)
java  YDataSet get_recordedData( long startTime, long endTime)
py    def get_recordedData( startTime, endTime)
cmd   YVoc target 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.

```
js function get_reportFrequency( )  
nodejs function get_reportFrequency( )  
php function get_reportFrequency( )  
cpp string get_reportFrequency( )  
m -(NSString*) reportFrequency  
pas function get_reportFrequency( ): string  
vb function get_reportFrequency( ) As String  
cs string get_reportFrequency( )  
java String get_reportFrequency( )  
py def get_reportFrequency( )  
cmd YVoc target 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.

```
js function get_resolution( )
nodejs function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YVoc target 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()  
voc→unit()voc.get\_unit()****YVoc**

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

js	function <b>get_unit( )</b>
node.js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) unit
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	<b>YVoc target get_unit</b>

**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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

js	function <b>isOnline()</b>
nodejs	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()**

YVoc

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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.

```

js   function loadCalibrationPoints( rawValues, refValues)
node.js function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)

m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function loadCalibrationPoints( var rawValues: TDoubleArray,
                           var refValues: TDoubleArray): LongInt

vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                           List<double> refValues)

java int loadCalibrationPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def loadCalibrationPoints( rawValues, refValues)
cmd  YVoc target loadCalibrationPoints rawValues refValues

```

### 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()****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

**js** `function load_async( msValidity, callback, context)`  
**nodejs** `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()`.

js	<code>function nextVoc()</code>
node.js	<code>function nextVoc()</code>
php	<code>function nextVoc()</code>
cpp	<code>YVoc * nextVoc()</code>
m	<code>-(YVoc*) nextVoc</code>
pas	<code>function nextVoc(): TYVoc</code>
vb	<code>function nextVoc() As YVoc</code>
cs	<code>YVoc nextVoc()</code>
java	<code>YVoc nextVoc()</code>
py	<code>def nextVoc()</code>

**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.

```
js function registerTimedReportCallback( callback)
node.js function registerTimedReportCallback( callback)
php function registerTimedReportCallback( $callback)
cpp int registerTimedReportCallback( YVocTimedReportCallback callback)
m -(int) registerTimedReportCallback : (YVocTimedReportCallback) callback
pas function registerTimedReportCallback( callback: TYVocTimedReportCallback): LongInt
vb function registerTimedReportCallback( ) As Integer
cs int registerTimedReportCallback( TimedReportCallback callback)
java int registerTimedReportCallback( TimedReportCallback callback)
py def registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

## voc→registerValueCallback() voc.registerValueCallback()

YVoc

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback( <b>callback</b> )
node.js	function registerValueCallback( <b>callback</b> )
php	function registerValueCallback( <b>\$callback</b> )
cpp	int registerValueCallback( YVocValueCallback <b>callback</b> )
m	-(int) registerValueCallback : (YVocValueCallback) <b>callback</b>
pas	function registerValueCallback( <b>callback</b> : TYVocValueCallback): LongInt
vb	function registerValueCallback( ) As Integer
cs	int registerValueCallback( ValueCallback <b>callback</b> )
java	int registerValueCallback( UpdateCallback <b>callback</b> )
py	def registerValueCallback( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**voc→set\_highestValue()**  
**voc→setHighestValue()voc.set\_highestValue()****YVoc**

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

```
js function set_highestValue( newval)
node.js function set_highestValue( newval)
php function set_highestValue( $newval)
cpp int set_highestValue( double newval)
m -(int) setHighestValue : (double) newval
pas function set_highestValue( newval: double): integer
vb function set_highestValue( ByVal newval As Double) As Integer
cs int set_highestValue( double newval)
java int set_highestValue( double newval)
py def set_highestValue( newval)
cmd YVoc target 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.

js	function <b>set_logFrequency( newval)</b>
node.js	function <b>set_logFrequency( newval)</b>
php	function <b>set_logFrequency( \$newval)</b>
cpp	int <b>set_logFrequency( const string&amp; newval)</b>
m	-(int) setLogFrequency : (NSString*) <b>newval</b>
pas	function <b>set_logFrequency( newval: string): integer</b>
vb	function <b>set_logFrequency( ByVal newval As String) As Integer</b>
cs	int <b>set_logFrequency( string newval)</b>
java	int <b>set_logFrequency( String newval)</b>
py	<b>def set_logFrequency( newval)</b>
cmd	<b>YVoc target set_logFrequency newval</b>

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()**  
**voc→setLogicalName()voc.set\_logicalName()****YVoc**

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
js function set_logicalName( newval)
node.js function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YVoc 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 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.

<b>js</b>	function <b>set_lowestValue( newval)</b>
<b>nodejs</b>	function <b>set_lowestValue( newval)</b>
<b>php</b>	function <b>set_lowestValue( \$newval)</b>
<b>cpp</b>	int <b>set_lowestValue( double newval)</b>
<b>m</b>	-(int) <b>setLowestValue : (double) newval</b>
<b>pas</b>	function <b>set_lowestValue( newval: double): integer</b>
<b>vb</b>	function <b>set_lowestValue( ByVal newval As Double) As Integer</b>
<b>cs</b>	int <b>set_lowestValue( double newval)</b>
<b>java</b>	int <b>set_lowestValue( double newval)</b>
<b>py</b>	def <b>set_lowestValue( newval)</b>
<b>cmd</b>	<b>YVoc target set_lowestValue newval</b>

**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.

```
js function set_reportFrequency( newval)
nodejs function set_reportFrequency( newval)
php function set_reportFrequency( $newval)
cpp int set_reportFrequency( const string& newval)
m -(int) setReportFrequency : (NSString*) newval
pas function set_reportFrequency( newval: string): integer
vb function set_reportFrequency( ByVal newval As String) As Integer
cs int set_reportFrequency( string newval)
java int set_reportFrequency( String newval)
py def set_reportFrequency( newval)
cmd YVoc target setReportFrequency 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.

js	function <b>set_resolution( newval)</b>
nodejs	function <b>set_resolution( newval)</b>
php	function <b>set_resolution( \$newval)</b>
cpp	int <b>set_resolution( double newval)</b>
m	-(int) <b>setResolution : (double) newval</b>
pas	function <b>set_resolution( newval: double): integer</b>
vb	function <b>set_resolution( ByVal newval As Double) As Integer</b>
cs	int <b>set_resolution( double newval)</b>
java	int <b>set_resolution( double newval)</b>
py	def <b>set_resolution( newval)</b>
cmd	<b>YVoc target set_resolution newval</b>

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set(userData)**  
**voc→setUserData()|voc.set(userData)****YVoc**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
js function set(userData( data)
nodejs function set(userData( data)
php function set(userData( $data)
cpp void set(userData( void* data)
m -(void) setUserData : (void*) data
pas procedure set(userData( data: Tobject)
vb procedure set(userData( ByVal data As Object)
cs void set(userData( object data)
java void set(userData( Object data)
py def set(userData( data)
```

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()****YVoc**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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()YVoltage.FindVoltage()**

Permet de retrouver un capteur de tension d'après un identifiant donné.

js	function <b>yFindVoltage( func)</b>
node.js	function <b>FindVoltage( func)</b>
php	function <b>yFindVoltage( \$func)</b>
cpp	YVoltage* <b>yFindVoltage( const string&amp; func)</b>
m	YVoltage* <b>yFindVoltage( NSString* func)</b>
pas	function <b>yFindVoltage( func: string): TYVoltage</b>
vb	function <b>yFindVoltage( ByVal func As String) As YVoltage</b>
cs	YVoltage <b>FindVoltage( string func)</b>
java	YVoltage <b>FindVoltage( String func)</b>
py	def <b>FindVoltage( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltage.isOnline()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de tension sans ambiguïté

**Retourne :**

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

**YVoltage.FirstVoltage()****YVoltage****yFirstVoltage() YVoltage.FirstVoltage()**

Commence l'énumération des capteurs de tension accessibles par la librairie.

js	function <b>yFirstVoltage( )</b>
node.js	function <b>FirstVoltage( )</b>
php	function <b>yFirstVoltage( )</b>
cpp	<b>YVoltage*</b> <b>yFirstVoltage( )</b>
m	<b>YVoltage*</b> <b>yFirstVoltage( )</b>
pas	function <b>yFirstVoltage( )</b> : TYVoltage
vb	function <b>yFirstVoltage( )</b> As <b>YVoltage</b>
cs	<b>YVoltage FirstVoltage( )</b>
java	<b>YVoltage FirstVoltage( )</b>
py	def <b>FirstVoltage( )</b>

Utiliser la fonction `YVoltage.nextVoltage( )` pour itérer sur les autres capteurs de tension.

**Retourne :**

un pointeur sur un objet `YVoltage`, correspondant à le premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

## voltage→calibrateFromPoints() 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.

```

js   function calibrateFromPoints( rawValues, refValues)
nodejs function calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
cpp   int calibrateFromPoints( vector<double> rawValues,
                               vector<double> refValues)

m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues

pas  function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt

vb   procedure calibrateFromPoints( )
cs    int calibrateFromPoints( List<double> rawValues,
                           List<double> refValues)

java int calibrateFromPoints( ArrayList<Double> rawValues,
                           ArrayList<Double> refValues)

py   def calibrateFromPoints( rawValues, refValues)
cmd  YVoltage target calibrateFromPoints rawValues refValues

```

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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

js	function <b>get_advertisedValue( )</b>
node.js	function <b>get_advertisedValue( )</b>
php	function <b>get_advertisedValue( )</b>
cpp	string <b>get_advertisedValue( )</b>
m	-(NSString*) <b>advertisedValue</b>
pas	function <b>get_advertisedValue( )</b> : string
vb	function <b>get_advertisedValue( )</b> As String
cs	string <b>get_advertisedValue( )</b>
java	String <b>get_advertisedValue( )</b>
py	def <b>get_advertisedValue( )</b>
cmd	YVoltage <b>target get_advertisedValue</b>

**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).

```
js function get_currentRawValue( )  
nodejs function get_currentRawValue( )  
php function get_currentRawValue( )  
cpp double get_currentRawValue( )  
m -(double) currentRawValue  
pas function get_currentRawValue( ): double  
vb function get_currentRawValue( ) As Double  
cs double get_currentRawValue( )  
java double get_currentRawValue( )  
py def get_currentRawValue( )  
cmd YVoltage target get_currentRawValue
```

**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.

**voltage→get\_currentValue()****YVoltage****voltage→currentValue()voltage.get\_currentValue()**

Retourne la valeur instantanée de la tension.

```
js function get_currentValue( )  
nodejs function get_currentValue( )  
php function get_currentValue( )  
cpp double get_currentValue( )  
m -(double) currentValue  
pas function get_currentValue( ): double  
vb function get_currentValue( ) As Double  
cs double get_currentValue( )  
java double get_currentValue( )  
py def get_currentValue( )  
cmd YVoltage target get_currentValue
```

**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.

```
js function get_errorMessage( )  
nodejs function get_errorMessage( )  
php function get_errorMessage( )  
cpp string get_errorMessage( )  
m -(NSString*) errorMessage  
pas function get_errorMessage( ):string  
vb function get_errorMessage( ) As String  
cs string get_errorMessage( )  
java String get_errorMessage( )  
py def get_errorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

**voltage→get\_friendlyName()****YVoltage****voltage→friendlyName()voltage.get\_friendlyName()**

Retourne un identifiant global du capteur de tension au format NOM\_MODULE.NOM\_FONCTION.

```
js function get_friendlyName( )  
node.js function get_friendlyName( )  
php function get_friendlyName( )  
cpp string get_friendlyName( )  
m -(NSString*) friendlyName  
cs string get_friendlyName( )  
java String get_friendlyName( )  
py def get_friendlyName( )
```

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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()**  
**voltage→functionId()voltage.get\_functionId()****YVoltage**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

```
js function get_functionId( )
node.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( )
py def get_functionId( )
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voltage→get\_hardwareId()****YVoltage****voltage→hardwareId()voltage.get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
nodejs	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

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.

```
js function get_highestValue( )  
node.js function get_highestValue( )  
php function get_highestValue( )  
cpp double get_highestValue( )  
m -(double) highestValue  
pas function get_highestValue( ): double  
vb function get_highestValue( ) As Double  
cs double get_highestValue( )  
java double get_highestValue( )  
py def get_highestValue( )  
cmd YVoltage target 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.

```
js function get_logFrequency( )
nodejs function get_logFrequency( )
php function get_logFrequency( )
cpp string get_logFrequency( )
m -(NSString*) logFrequency
pas function get_logFrequency( ): string
vb function get_logFrequency( ) As String
cs string get_logFrequency( )
java String get_logFrequency( )
py def get_logFrequency( )
cmd YVoltage target get_logFrequency
```

**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.

js	function <b>get_logicalName( )</b>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( ): string</b>
vb	function <b>get_logicalName( ) As String</b>
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	def <b>get_logicalName( )</b>
cmd	<b>YVoltage target get_logicalName</b>

**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()**  
**voltage→lowestValue()voltage.get\_lowestValue()****YVoltage**

Retourne la valeur minimale observée pour la tension.

```
js function get_lowestValue( )  
nodejs function get_lowestValue( )  
php function get_lowestValue( )  
cpp double get_lowestValue( )  
m -(double) lowestValue  
pas function get_lowestValue( ): double  
vb function get_lowestValue( ) As Double  
cs double get_lowestValue( )  
java double get_lowestValue( )  
py def get_lowestValue( )  
cmd YVoltage target 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()**  
**voltage→module()voltage.get\_module()****YVoltage**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YVoltage**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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.

## voltage→get\_recordedData() YVoltage

## voltage→recordedData() voltage.get\_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```

js function get_recordedData( startTime, endTime)
nodejs function get_recordedData( startTime, endTime)
php function get_recordedData( $startTime, $endTime)
cpp YDataSet get_recordedData( s64 startTime, s64 endTime)
m -(YDataSet*) recordedData : (s64) startTime
: (s64) endTime
pas function get_recordedData( startTime: int64, endTime: int64): TYDataSet
vb function get_recordedData( ) As YDataSet
cs YDataSet get_recordedData( long startTime, long endTime)
java YDataSet get_recordedData( long startTime, long endTime)
py def get_recordedData( startTime, endTime)
cmd YVoltage target 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.

```
js function get_reportFrequency( )
nodejs function get_reportFrequency( )
php function get_reportFrequency( )
cpp string get_reportFrequency( )
m -(NSString*) reportFrequency
pas function get_reportFrequency( ): string
vb function get_reportFrequency( ) As String
cs string get_reportFrequency( )
java String get_reportFrequency( )
py def get_reportFrequency( )
cmd YVoltage target get_reportFrequency
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**voltage→get\_resolution()** **YVoltage**  
**voltage→resolution()voltage.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
js function get_resolution( )
node.js function get_resolution( )
php function get_resolution( )
cpp double get_resolution( )
m -(double) resolution
pas function get_resolution( ): double
vb function get_resolution( ) As Double
cs double get_resolution( )
java double get_resolution( )
py def get_resolution( )
cmd YVoltage target 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.

```
js function get_unit( )
nodejs function get_unit( )
php function get_unit( )
cpp string get_unit( )
m -(NSString*) unit
pas function get_unit( ): string
vb function get_unit( ) As String
cs string get_unit( )
java String get_unit( )
py def get_unit( )
cmd YVoltage 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.

**voltage→get(userData)**  
**voltage→userData(voltage.get(userData))****YVoltage**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData) {  
m -(void*) userData  
pas function get(userData): Tobject  
vb function get(userData) As Object  
cs object get(userData)  
java Object get(userData)  
py def get(userData)
```

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.

js	function <b>isOnline( )</b>
node.js	function <b>isOnline( )</b>
php	function <b>isOnline( )</b>
cpp	bool <b>isOnline( )</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline( )</b> : boolean
vb	function <b>isOnline( )</b> As Boolean
cs	bool <b>isOnline( )</b>
java	boolean <b>isOnline( )</b>
py	<b>def isOnline( )</b>

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()

YVoltage

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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() voltage.loadCalibrationPoints()

YVoltage

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```

js   function loadCalibrationPoints( rawValues, refValues)
nodejs function loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
cpp   int loadCalibrationPoints( vector<double>& rawValues,
                                vector<double>& refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                           : (NSMutableArray*) refValues
pas   function loadCalibrationPoints( var rawValues: TDoubleArray,
                                      var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( )
cs    int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
py    def loadCalibrationPoints( rawValues, refValues)
cmd   YVoltage target loadCalibrationPoints rawValues refValues

```

### 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()

## YVoltage

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextVoltage()</b>
nodejs	function <b>nextVoltage()</b>
php	function <b>nextVoltage()</b>
cpp	<b>YVoltage *</b> <b>nextVoltage()</b>
m	<b>-(YVoltage*) nextVoltage</b>
pas	function <b>nextVoltage()</b> : TYVoltage
vb	function <b>nextVoltage()</b> As YVoltage
cs	<b>YVoltage nextVoltage()</b>
java	<b>YVoltage nextVoltage()</b>
py	def <b>nextVoltage()</b>

**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.

js	function registerTimedReportCallback( <b>callback</b> )
node.js	function registerTimedReportCallback( <b>callback</b> )
php	function registerTimedReportCallback( <b>\$callback</b> )
cpp	int registerTimedReportCallback( YVoltageTimedReportCallback <b>callback</b> )
m	-(int) registerTimedReportCallback : (YVoltageTimedReportCallback) <b>callback</b>
pas	function registerTimedReportCallback( <b>callback</b> : TYVoltageTimedReportCallback): LongInt
vb	function registerTimedReportCallback( ) As Integer
cs	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
java	int registerTimedReportCallback( TimedReportCallback <b>callback</b> )
py	def registerTimedReportCallback( <b>callback</b> )

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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YVoltageValueCallback callback)
m -(int) registerValueCallback : (YVoltageValueCallback) callback
pas function registerValueCallback( callback: TYVoltageValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

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.

js	function <b>set_highestValue( newval)</b>
node.js	function <b>set_highestValue( newval)</b>
php	function <b>set_highestValue( \$newval)</b>
cpp	int <b>set_highestValue( double newval)</b>
m	-(int) setHighestValue : (double) <b>newval</b>
pas	function <b>set_highestValue( newval: double): integer</b>
vb	function <b>set_highestValue( ByVal newval As Double) As Integer</b>
cs	int <b>set_highestValue( double newval)</b>
java	int <b>set_highestValue( double newval)</b>
py	def <b>set_highestValue( newval)</b>
cmd	YVoltage <b>target set_highestValue newval</b>

**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.

```
js function set_logFrequency( newval)
nodejs function set_logFrequency( newval)
php function set_logFrequency( $newval)
cpp int set_logFrequency( const string& newval)
m -(int) setLogFrequency : (NSString*) newval
pas function set_logFrequency( newval: string): integer
vb function set_logFrequency( ByVal newval As String) As Integer
cs int set_logFrequency( string newval)
java int set_logFrequency( String newval)
py def set_logFrequency( newval)
cmd YVoltage target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## voltage→set\_logicalName() voltage→setLogicalName()voltage.set\_logicalName()

YVoltage

Modifie le nom logique du capteur de tension.

js	function <b>set_logicalName( newval)</b>
nodejs	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	<b>YVoltage target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

### Paramètres :

**newval** une chaîne de caractères représentant le nom logique 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() voltage→setLowestValue()voltage.set\_lowestValue()

YVoltage

Modifie la mémoire de valeur minimale observée pour la tension.

```
js function set_lowestValue( newval)
node.js function set_lowestValue( newval)
php function set_lowestValue( $newval)
cpp int set_lowestValue( double newval)
m -(int) setLowestValue : (double) newval
pas function set_lowestValue( newval: double): integer
vb function set_lowestValue( ByVal newval As Double) As Integer
cs int set_lowestValue( double newval)
java int set_lowestValue( double newval)
py def set_lowestValue( newval)
cmd YVoltage target 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.

<code>js</code>	<code>function set_reportFrequency( newval)</code>
<code>node.js</code>	<code>function set_reportFrequency( newval)</code>
<code>php</code>	<code>function set_reportFrequency( \$newval)</code>
<code>cpp</code>	<code>int set_reportFrequency( const string&amp; newval)</code>
<code>m</code>	<code>-(int) setReportFrequency : (NSString*) newval</code>
<code>pas</code>	<code>function set_reportFrequency( newval: string): integer</code>
<code>vb</code>	<code>function set_reportFrequency( ByVal newval As String) As Integer</code>
<code>cs</code>	<code>int set_reportFrequency( string newval)</code>
<code>java</code>	<code>int set_reportFrequency( String newval)</code>
<code>py</code>	<code>def set_reportFrequency( newval)</code>
<code>cmd</code>	<code>YVoltage target set_reportFrequency newval</code>

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.

```
js function set_resolution( newval)
node.js function set_resolution( newval)
php function set_resolution( $newval)
cpp int set_resolution( double newval)
m -(int) setResolution : (double) newval
pas function set_resolution( newval: double): integer
vb function set_resolution( ByVal newval As Double) As Integer
cs int set_resolution( double newval)
java int set_resolution( double newval)
py def set_resolution( newval)
cmd YVoltage target 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).

js	function <b>set(userData)</b>
node.js	function <b>set(userData)</b>
php	function <b>set(userData \$data)</b>
cpp	void <b>set(userData void* data)</b>
m	- <b>(void) set(userData : (void*) data)</b>
pas	procedure <b>set(userData Tobject data)</b>
vb	procedure <b>set(userData ByVal data As Object)</b>
cs	void <b>set(userData object data)</b>
java	void <b>set(userData Object data)</b>
py	def <b>set(userData data)</b>

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()

YVoltage

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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****YVSource.FindVSource()YVSource.FindVSource()**

Permet de retrouver une source de tension d'après un identifiant donné.

<b>js</b>	<code>function yFindVSource( func)</code>
<b>php</b>	<code>function yFindVSource( \$func)</code>
<b>cpp</b>	<code>YVSource* yFindVSource( const string&amp; func)</code>
<b>m</b>	<code>YVSource* yFindVSource( NSString* func)</code>
<b>pas</b>	<code>function yFindVSource( func: string): TYVSource</code>
<b>vb</b>	<code>function yFindVSource( ByVal func As String) As YVSource</code>
<b>cs</b>	<code>YVSource FindVSource( string func)</code>
<b>java</b>	<code>YVSource FindVSource( String func)</code>
<b>py</b>	<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'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

**Retourne :**

un objet de classe `YVSource` qui permet ensuite de contrôler la source de tension.

**yFirstVSource()** —**YVSource****YVSource.FirstVSource()YVSource.FirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

js	function <b>yFirstVSource()</b>
php	function <b>yFirstVSource()</b>
cpp	YVSource* <b>yFirstVSource()</b>
m	YVSource* <b>yFirstVSource()</b>
pas	function <b>yFirstVSource()</b> : TYVSource
vb	function <b>yFirstVSource()</b> As YVSource
cs	<b>YVSource FirstVSource()</b>
java	<b>YVSource FirstVSource()</b>
py	def <b>FirstVSource()</b>

Utiliser la fonction `YVSource.nextVSource()` pour itérer sur les autres sources de tension.

**Retourne :**

un pointeur sur un objet `YVSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de sources de tension disponibles.

**vsource→describe()|vsource.describe()****YVSource**

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	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )

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 fonction (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**vsOURCE→get\_advertisedValue()**  
**vsOURCE→advertisedValue()**  
**vsOURCE.get\_advertisedValue()**

**YVSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
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.

**vsources->getErrorMessage()**  
**vsources->errorMessage()**  
**vsources.getErrorMessage()****YVSource**

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

**vsouce→get\_errorType()** **YVSource**  
**vsouce→errorType()vsouce.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
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

**vsource→get\_extPowerFailure()**  
**vsource→extPowerFailure()**  
**vsource.get\_extPowerFailure()**

**YVSource**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

```
js function get_extPowerFailure( )
php function get_extPowerFailure( )
cpp Y_EXTPOWERFAILURE_enum get_extPowerFailure( )
m -(Y_EXTPOWERFAILURE_enum) extPowerFailure
pas function get_extPowerFailure( ): Integer
vb function get_extPowerFailure( ) As Integer
cs int get_extPowerFailure( )
java int get_extPowerFailure( )
py def get_extPowerFailure( )
cmd YVSource target get_extPowerFailure
```

**Retourne :**

soit Y\_EXTPOWERFAILURE\_FALSE, soit Y\_EXTPOWERFAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_EXTPOWERFAILURE\_INVALID.

**vsources->get\_failure()**  
**vsources->failure() vsources.get\_failure()****YVSource**

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.

```
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.

**vsouce→get\_functionDescriptor()  
vsouce→functionDescriptor()  
vsouce.get\_vsouceDescriptor()****YVSource**

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 renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**vsouce→get\_functionId()****YVSource****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.

**vsouce→get\_hardwareId()****YVSource****vsouce→hardwareId()vsouce.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 RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: RELAYLO1-123456.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.

```
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.

**vsouce→get\_module()**  
**vsouce→module()vsouce.get\_module()****YVSource**

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()****YVSource**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`js 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.

**vsOURCE→get\_overCurrent()** **YVSource**  
**vsOURCE→overCurrent()vsOURCE.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.

```
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.

```
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 rapport à la tension demandée demandée.

```
js function get_regulationFailure( )
php function get_regulationFailure( )
cpp Y_REGULATIONFAILURE_enum get_regulationFailure( )
m -(Y_REGULATIONFAILURE_enum) regulationFailure
pas function get_regulationFailure( ): Integer
vb function get_regulationFailure( ) As Integer
cs int get_regulationFailure( )
java int get_regulationFailure( )
py def get_regulationFailure( )
cmd YVSource target get_regulationFailure
```

**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.

js	function <b>get_unit( )</b>
php	function <b>get_unit( )</b>
cpp	string <b>get_unit( )</b>
m	-(NSString*) <b>unit</b>
pas	function <b>get_unit( )</b> : string
vb	function <b>get_unit( )</b> As String
cs	string <b>get_unit( )</b>
java	String <b>get_unit( )</b>
py	def <b>get_unit( )</b>
cmd	YVSource <b>target get_unit</b>

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**vsource→get(userData)****YVSource****vsource→userData()vsource.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

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

```
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.

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()****YVSource**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

**js** `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.

```
js function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsources→load\_async()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

`js 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()`.

js	function <b>nextVSource()</b>
php	function <b>nextVSource()</b>
cpp	YVSource * <b>nextVSource()</b>
m	-(YVSource*) <b>nextVSource</b>
pas	function <b>nextVSource()</b> : TYVSource
vb	function <b>nextVSource()</b> As YVSource
cs	YVSource <b>nextVSource()</b>
java	YVSource <b>nextVSource()</b>
py	def <b>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.

```
js function pulse( voltage, ms_duration)
php function pulse( $voltage, $ms_duration)
cpp int pulse( int voltage, int ms_duration)
m -(int) pulse : (int) voltage : (int) ms_duration
pas function pulse( voltage: integer, ms_duration: integer): integer
vb function pulse( ByVal voltage As Integer,
                   ByVal ms_duration As Integer) As Integer
cs int pulse( int voltage, int ms_duration)
java int pulse( int voltage, int ms_duration)
py def pulse( voltage, ms_duration)
cmd YVSource target pulse voltage ms_duration
```

**Paramètres :**

**voltage** tension demandée, en millivolts  
**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **vsources->registerValueCallback()**

### **vsources.registerValueCallback()**

**YVSource**

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.

**vsOURCE→set\_logicalName()**  
**vsOURCE→setLogicalName()**  
**vsOURCE.set\_logicalName()**

**YVSource**

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).

js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**vsourceset\_voltage()** **YVSource**  
**vsourcesetVoltage()|vsourceset\_voltage()**

Règle la tension de sortie du module (en millivolts).

```
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.

```
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()****YVSource**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**js** `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()**

## **YWakeUpMonitor.FindWakeUpMonitor()**

**YWakeUpMonitor**

Permet de retrouver un moniteur d'après un identifiant donné.

<b>js</b>	function <b>yFindWakeUpMonitor( func)</b>
<b>nodejs</b>	function <b>FindWakeUpMonitor( func)</b>
<b>php</b>	function <b>yFindWakeUpMonitor( \$func)</b>
<b>cpp</b>	YWakeUpMonitor* <b>yFindWakeUpMonitor( const string&amp; func)</b>
<b>m</b>	YWakeUpMonitor* <b>yFindWakeUpMonitor( NSString* func)</b>
<b>pas</b>	function <b>yFindWakeUpMonitor( func: string): TYWakeUpMonitor</b>
<b>vb</b>	function <b>yFindWakeUpMonitor( ByVal func As String) As YWakeUpMonitor</b>
<b>cs</b>	YWakeUpMonitor <b>FindWakeUpMonitor( string func)</b>
<b>java</b>	YWakeUpMonitor <b>FindWakeUpMonitor( String func)</b>
<b>py</b>	def <b>FindWakeUpMonitor( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.isOnLine()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

#### **Paramètres :**

**func** une chaîne de caractères qui référence le moniteur sans ambiguïté

#### **Retourne :**

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

## YWakeUpMonitor.FirstWakeUpMonitor() yFirstWakeUpMonitor() YWakeUpMonitor.FirstWakeUpMonitor()

## YWakeUpMonitor

Commence l'énumération des Moniteurs accessibles par la librairie.

```
js    function yFirstWakeUpMonitor( )
node.js function FirstWakeUpMonitor( )
php   function yFirstWakeUpMonitor( )
cpp   YWakeUpMonitor* yFirstWakeUpMonitor( )
m     YWakeUpMonitor* yFirstWakeUpMonitor( )
pas   function yFirstWakeUpMonitor( ): TYWakeUpMonitor
vb    function yFirstWakeUpMonitor( ) As YWakeUpMonitor
cs    YWakeUpMonitor FirstWakeUpMonitor( )
java  YWakeUpMonitor FirstWakeUpMonitor( )
py    def FirstWakeUpMonitor( )
```

Utiliser la fonction `YWakeUpMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

### Retourne :

un pointeur sur un objet `YWakeUpMonitor`, correspondant à le 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	- <b>(NSString*</b> ) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le moniteur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

wakeupmonitor→get\_advertisedValue()  
wakeupmonitor→advertisedValue()  
wakeupmonitor.get\_advertisedValue()

YWakeUpMonitor

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
node.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 YWakeUpMonitor target get_advertisedValue
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wakeupmonitor→getErrorMessage()**  
**wakeupmonitor→errorMessage()**  
**wakeupmonitor.getErrorMessage()****YWakeUpMonitor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

js	function getErrorMessage( )
nodejs	function getErrorMessage( )
php	function getErrorMessage( )
cpp	string getErrorMessage( )
m	-(NSString*) errorMessage
pas	function getErrorMessage( ): string
vb	function getErrorMessage( ) As String
cs	string getErrorMessage( )
java	String getErrorMessage( )
py	def getErrorMessage( )

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.

js	function <b>get_errorType( )</b>
nodejs	function <b>get_errorType( )</b>
php	function <b>get_errorType( )</b>
cpp	<b>YRETCODE get_errorType( )</b>
pas	function <b>get_errorType( )</b> : YRETCODE
vb	function <b>get_errorType( )</b> As YRETCODE
cs	<b>YRETCODE get_errorType( )</b>
java	<b>int get_errorType( )</b>
py	<b>def get_errorType( )</b>

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

**wakeupmonitor→get\_friendlyName()**  
**wakeupmonitor→friendlyName()**  
**wakeupmonitor.get\_friendlyName()****YWakeUpMonitor**

Retourne un identifiant global du moniteur au format NOM\_MODULE.NOM\_FONCTION.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function <b>get_functionDescriptor()</b>
node.js	function <b>get_functionDescriptor()</b>
php	function <b>get_functionDescriptor()</b>
cpp	<b>YFUN_DESCR get_functionDescriptor()</b>
m	-(YFUN_DESCR) functionDescriptor
pas	function <b>get_functionDescriptor()</b> : YFUN_DESCR
vb	function <b>get_functionDescriptor()</b> As YFUN_DESCR
cs	<b>YFUN_DESCR get_functionDescriptor()</b>
java	<b>String get_functionDescriptor()</b>
py	<b>def get_functionDescriptor()</b>

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.

js	function get_functionId( )
nodejs	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( )
py	def get_functionId( )

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le moniteur (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

**wakeupmonitor→get\_hardwareId()**  
**wakeupmonitor→hardwareId()**  
**wakeupmonitor.get\_hardwareId()****YWakeUpMonitor**

Retourne l'identifiant matériel unique du moniteur au format SERIAL.FUNCTIONID.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	<b>String get_hardwareId( )</b>
py	<b>def get_hardwareId( )</b>

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.

js	function get_logicalName( )
nodejs	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YWakeUpMonitor target get_logicalName

**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.

js	function <b>get_module</b> ( )
nodejs	function <b>get_module</b> ( )
php	function <b>get_module</b> ( )
cpp	<code>YModule * get_module( )</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module</b> ( ): TYModule
vb	function <b>get_module</b> ( ) As YModule
cs	<code>YModule get_module( )</code>
java	<code>YModule get_module( )</code>
py	<code>def get_module( )</code>

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()****YWakeUpMonitor**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
node.js 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()  
wakeupmonitor→nextWakeUp()  
wakeupmonitor.get\_nextWakeUp()

YWakeUpMonitor

Retourne la prochaine date/heure de réveil agendée (format UNIX)

js	function get_nextWakeUp( )
node.js	function get_nextWakeUp( )
php	function get_nextWakeUp( )
cpp	s64 get_nextWakeUp( )
m	-(s64) nextWakeUp
pas	function get_nextWakeUp( ): int64
vb	function get_nextWakeUp( ) As Long
cs	long get_nextWakeUp( )
java	long get_nextWakeUp( )
py	def get_nextWakeUp( )

**Retourne :**

un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTWAKEUP\_INVALID.

**wakeupmonitor→get\_powerDuration()****YWakeUpMonitor****wakeupmonitor→powerDuration()****wakeupmonitor.get\_powerDuration()**

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
js function get_powerDuration( )
nodejs function get_powerDuration( )
php function get_powerDuration( )
cpp int get_powerDuration( )
m -(int) powerDuration
pas function get_powerDuration( ): LongInt
vb function get_powerDuration( ) As Integer
cs int get_powerDuration( )
java int get_powerDuration( )
py def get_powerDuration( )
cmd YWakeUpMonitor target 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.

js	function get_sleepCountdown( )
node.js	function get_sleepCountdown( )
php	function get_sleepCountdown( )
cpp	int get_sleepCountdown( )
m	-(int) sleepCountdown
pas	function get_sleepCountdown( ): LongInt
vb	function get_sleepCountdown( ) As Integer
cs	int get_sleepCountdown( )
java	int get_sleepCountdown( )
py	def get_sleepCountdown( )
cmd	YWakeUpMonitor target get_sleepCountdown

**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).

js	function get(userData) {
nodejs	function get(userData) {
php	function get(userData) {
cpp	void * get(userData) {
m	-(void*) userData
pas	function get(userData): Tobject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

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.

js	function <b>get_wakeUpReason( )</b>
node.js	function <b>get_wakeUpReason( )</b>
php	function <b>get_wakeUpReason( )</b>
cpp	Y_WAKEUPREASON_enum <b>get_wakeUpReason( )</b>
m	-(Y_WAKEUPREASON_enum) <b>wakeUpReason</b>
pas	function <b>get_wakeUpReason( )</b> : Integer
vb	function <b>get_wakeUpReason( )</b> As Integer
cs	int <b>get_wakeUpReason( )</b>
java	int <b>get_wakeUpReason( )</b>
py	def <b>get_wakeUpReason( )</b>
cmd	<b>YWakeUpMonitor target get_wakeUpReason</b>

**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

```
js function get_wakeUpState( )  
nodejs function get_wakeUpState( )  
php function get_wakeUpState( )  
cpp Y_WAKEUPSTATE_enum get_wakeUpState( )  
m -(Y_WAKEUPSTATE_enum) wakeUpState  
pas function get_wakeUpState( ): Integer  
vb function get_wakeUpState( ) As Integer  
cs int get_wakeUpState( )  
java int get_wakeUpState( )  
py def get_wakeUpState( )
```

**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.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()

## YWakeUpMonitor

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()****YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

<code>js</code>	<code>function nextWakeUpMonitor( )</code>
<code>nodejs</code>	<code>function nextWakeUpMonitor( )</code>
<code>php</code>	<code>function nextWakeUpMonitor( )</code>
<code>cpp</code>	<code>YWakeUpMonitor * nextWakeUpMonitor( )</code>
<code>m</code>	<code>-(YWakeUpMonitor*) nextWakeUpMonitor</code>
<code>pas</code>	<code>function nextWakeUpMonitor( ): TYWakeUpMonitor</code>
<code>vb</code>	<code>function nextWakeUpMonitor( ) As YWakeUpMonitor</code>
<code>cs</code>	<code>YWakeUpMonitor nextWakeUpMonitor( )</code>
<code>java</code>	<code>YWakeUpMonitor nextWakeUpMonitor( )</code>
<code>py</code>	<code>def nextWakeUpMonitor( )</code>

**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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YWakeUpMonitorValueCallback callback)
m -(int) registerValueCallback : (YWakeUpMonitorValueCallback) callback
pas function registerValueCallback( callback: TYWakeUpMonitorValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

js	function resetSleepCountDown( )
node.js	function resetSleepCountDown( )
php	function resetSleepCountDown( )
cpp	int resetSleepCountDown( )
m	-(int) resetSleepCountDown
pas	function resetSleepCountDown( ): LongInt
vb	function resetSleepCountDown( ) As Integer
cs	int resetSleepCountDown( )
java	int resetSleepCountDown( )
py	def resetSleepCountDown( )
cmd	YWakeUpMonitor target 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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YWakeUpMonitor 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 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.

js	function <b>set_nextWakeUp( newval)</b>
node.js	function <b>set_nextWakeUp( newval)</b>
php	function <b>set_nextWakeUp( \$newval)</b>
cpp	int <b>set_nextWakeUp( s64 newval)</b>
m	-(int) <b>setNextWakeUp : (s64) newval</b>
pas	function <b>set_nextWakeUp( newval: int64): integer</b>
vb	function <b>set_nextWakeUp( ByVal newval As Long) As Integer</b>
cs	int <b>set_nextWakeUp( long newval)</b>
java	int <b>set_nextWakeUp( long newval)</b>
py	def <b>set_nextWakeUp( newval)</b>
cmd	<b>YWakeUpMonitor target set_nextWakeUp newval</b>

**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.

js	function set_powerDuration( newval)
nodejs	function set_powerDuration( newval)
php	function set_powerDuration( \$newval)
cpp	int set_powerDuration( int newval)
m	-(int) setPowerDuration : (int) newval
pas	function set_powerDuration( newval: LongInt): integer
vb	function set_powerDuration( ByVal newval As Integer) As Integer
cs	int set_powerDuration( int newval)
java	int set_powerDuration( int newval)
py	def set_powerDuration( newval)
cmd	YWakeUpMonitor target 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 .

```
js function set_sleepCountdown( newval)
nodejs function set_sleepCountdown( newval)
php function set_sleepCountdown( $newval)
cpp int set_sleepCountdown( int newval)
m -(int) setSleepCountdown : (int) newval
pas function set_sleepCountdown( newval: LongInt): integer
vb function set_sleepCountdown( ByVal newval As Integer) As Integer
cs int set_sleepCountdown( int newval)
java int set_sleepCountdown( int newval)
py def set_sleepCountdown( newval)
cmd YWakeUpMonitor target set_sleepCountdown newval
```

**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).

```
js function set(userData) {  
nodejs function setUserData( data)  
php function setUserData( $data)  
cpp void setUserData( void* data)  
m -(void) setUserData : (void*) data  
pas procedure set(userData: Tobject)  
vb procedure setUserData( ByVal data As Object)  
cs void set(userData: object data)  
java void setUserData( Object data)  
py def set(userData: data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**wakeupmonitor→sleep()wakeupmonitor.sleep()****YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

js	function sleep( secBeforeSleep)
nodejs	function sleep( secBeforeSleep)
php	function sleep( \$secBeforeSleep)
cpp	int sleep( int secBeforeSleep)
m	-(int) sleep : (int) secBeforeSleep
pas	function sleep( secBeforeSleep: LongInt): LongInt
vb	function sleep( ) As Integer
cs	int sleep( int secBeforeSleep)
java	int sleep( int secBeforeSleep)
py	def sleep( secBeforeSleep)
cmd	YWakeUpMonitor target sleep secBeforeSleep

**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.

```

js   function sleepFor( secUntilWakeUp, secBeforeSleep)
nodejs function sleepFor( secUntilWakeUp, secBeforeSleep)
php  function sleepFor( $secUntilWakeUp, $secBeforeSleep)
cpp   int sleepFor( int secUntilWakeUp, int secBeforeSleep)
m    -(int) sleepFor : (int) secUntilWakeUp : (int) secBeforeSleep
pas   function sleepFor( secUntilWakeUp: LongInt,
                        secBeforeSleep: LongInt): LongInt
vb    function sleepFor( ) As Integer
cs    int sleepFor( int secUntilWakeUp, int secBeforeSleep)
java  int sleepFor( int secUntilWakeUp, int secBeforeSleep)
py    def sleepFor( secUntilWakeUp, secBeforeSleep)
cmd   YWakeUpMonitor target 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.

```

js   function sleepUntil( wakeUpTime, secBeforeSleep)
node.js function sleepUntil( wakeUpTime, secBeforeSleep)
php  function sleepUntil( $wakeUpTime, $secBeforeSleep)
cpp   int sleepUntil( int wakeUpTime, int secBeforeSleep)
m    -(int) sleepUntil : (int) wakeUpTime : (int) secBeforeSleep
pas   function sleepUntil( wakeUpTime: LongInt,
                           secBeforeSleep: LongInt): LongInt

vb   function sleepUntil( ) As Integer
cs    int sleepUntil( int wakeUpTime, int secBeforeSleep)
java  int sleepUntil( int wakeUpTime, int secBeforeSleep)
py    def sleepUntil( wakeUpTime, secBeforeSleep)
cmd   YWakeUpMonitor target 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()

## YWakeUpMonitor

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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→wakeUp()****YWakeUpMonitor**

Force un réveil.

js	function <b>wakeUp( )</b>
node.js	function <b>wakeUp( )</b>
php	function <b>wakeUp( )</b>
cpp	int <b>wakeUp( )</b>
m	- <b>(int) wakeUp</b>
pas	function <b>wakeUp( ): LongInt</b>
vb	function <b>wakeUp( ) As Integer</b>
cs	int <b>wakeUp( )</b>
java	int <b>wakeUp( )</b>
py	def <b>wakeUp( )</b>
cmd	YWakeUpMonitor <b>target wakeUp</b>

## 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()****YWakeUpSchedule****yFindWakeUpSchedule()****YWakeUpSchedule.FindWakeUpSchedule()**

Permet de retrouver un réveil agendé d'après un identifiant donné.

js	function <b>yFindWakeUpSchedule( func)</b>
node.js	function <b>FindWakeUpSchedule( func)</b>
php	function <b>yFindWakeUpSchedule( \$func)</b>
cpp	<b>YWakeUpSchedule*</b> <b>yFindWakeUpSchedule( const string&amp; func)</b>
m	<b>YWakeUpSchedule*</b> <b>yFindWakeUpSchedule( NSString* func)</b>
pas	function <b>yFindWakeUpSchedule( func: string): TYWakeUpSchedule</b>
vb	function <b>yFindWakeUpSchedule( ByVal func As String) As YWakeUpSchedule</b>
cs	<b>YWakeUpSchedule</b> <b>FindWakeUpSchedule( string func)</b>
java	<b>YWakeUpSchedule</b> <b>FindWakeUpSchedule( String func)</b>
py	def <b>FindWakeUpSchedule( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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()****YWakeUpSchedule****yFirstWakeUpSchedule()****YWakeUpSchedule.FirstWakeUpSchedule()**

Commence l'énumération des réveils agendés accessibles par la librairie.

js	function <b>yFirstWakeUpSchedule( )</b>
nodejs	function <b>FirstWakeUpSchedule( )</b>
php	function <b>yFirstWakeUpSchedule( )</b>
cpp	YWakeUpSchedule* <b>yFirstWakeUpSchedule( )</b>
m	YWakeUpSchedule* <b>yFirstWakeUpSchedule( )</b>
pas	function <b>yFirstWakeUpSchedule( )</b> : TYWakeUpSchedule
vb	function <b>yFirstWakeUpSchedule( )</b> As YWakeUpSchedule
cs	YWakeUpSchedule <b>FirstWakeUpSchedule( )</b>
java	YWakeUpSchedule <b>FirstWakeUpSchedule( )</b>
py	def <b>FirstWakeUpSchedule( )</b>

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule()` pour itérer sur les autres réveils agendés.

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule`, correspondant à le 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le réveil agendé (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

wakeupschedule→get\_advertisedValue()  
wakeupschedule→advertisedValue()  
wakeupschedule.get\_advertisedValue()

YWakeUpSchedule

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YWakeUpSchedule target 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→getErrorMessage()  
wakeupschedule→errorMessage()  
wakeupschedule.getErrorMessage()****YWakeUpSchedule**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
js function getErrorMessage( )
nodejs function getErrorMessage( )
php function getErrorMessage( )
cpp string getErrorMessage( )
m -(NSString*) errorMessage
pas function getErrorMessage( ): string
vb function getErrorMessage( ) As String
cs string getErrorMessage( )
java String getErrorMessage( )
py def getErrorMessage( )
```

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.

js	function <b>get_errorType()</b>
nodejs	function <b>get_errorType()</b>
php	function <b>get_errorType()</b>
cpp	YRETCODE <b>get_errorType()</b>
pas	function <b>get_errorType()</b> : YRETCODE
vb	function <b>get_errorType()</b> As YRETCODE
cs	YRETCODE <b>get_errorType()</b>
java	int <b>get_errorType()</b>
py	def <b>get_errorType()</b>

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.

js	function get_friendlyName( )
node.js	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

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..

```
js    function get_hours( )  
nodejs function get_hours( )  
php   function get_hours( )  
cpp   int get_hours( )  
m     -(int) hours  
pas   function get_hours( ): LongInt  
vb    function get_hours( ) As Integer  
cs    int get_hours( )  
java  int get_hours( )  
py    def get_hours( )  
cmd   YWakeUpSchedule target get_hours
```

**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.

js	function get_logicalName( )
nodejs	function get_logicalName( )
php	function get_logicalName( )
cpp	string get_logicalName( )
m	-(NSString*) logicalName
pas	function get_logicalName( ): string
vb	function get_logicalName( ) As String
cs	string get_logicalName( )
java	String get_logicalName( )
py	def get_logicalName( )
cmd	YWakeUpSchedule target 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.

js	function <b>get_minutes()</b>
node.js	function <b>get_minutes()</b>
php	function <b>get_minutes()</b>
cpp	s64 <b>get_minutes()</b>
m	-(s64) minutes
pas	function <b>get_minutes()</b> : int64
vb	function <b>get_minutes()</b> As Long
cs	long <b>get_minutes()</b>
java	long <b>get_minutes()</b>
py	def <b>get_minutes()</b>
cmd	YWakeUpSchedule <b>target get_minutes</b>

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.

```
js function get_minutesA( )
nodejs function get_minutesA( )
php function get_minutesA( )
cpp int get_minutesA( )
m -(int) minutesA
pas function get_minutesA( ): LongInt
vb function get_minutesA( ) As Integer
cs int get_minutesA( )
java int get_minutesA( )
py def get_minutesA( )
cmd YWakeUpSchedule target 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.

js	function get_minutesB( )
node.js	function get_minutesB( )
php	function get_minutesB( )
cpp	int get_minutesB( )
m	-(int) minutesB
pas	function get_minutesB( ): LongInt
vb	function get_minutesB( ) As Integer
cs	int get_minutesB( )
java	int get_minutesB( )
py	def get_minutesB( )
cmd	YWakeUpSchedule target get_minutesB

**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.

js	function <b>get_module()</b>
nodejs	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As <code>YModule</code>
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()****YWakeUpSchedule**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs 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..

js	function get_monthDays( )
nodejs	function get_monthDays( )
php	function get_monthDays( )
cpp	int get_monthDays( )
m	-(int) monthDays
pas	function get_monthDays( ): LongInt
vb	function get_monthDays( ) As Integer
cs	int get_monthDays( )
java	int get_monthDays( )
py	def get_monthDays( )
cmd	YWakeUpSchedule target get_monthDays

**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..

js	function <b>get_months()</b>
nodejs	function <b>get_months()</b>
php	function <b>get_months()</b>
cpp	int <b>get_months()</b>
m	-(int) months
pas	function <b>get_months()</b> : LongInt
vb	function <b>get_months()</b> As Integer
cs	int <b>get_months()</b>
java	int <b>get_months()</b>
py	def <b>get_months()</b>
cmd	YWakeUpSchedule <b>target get_months</b>

**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

```
js function get_nextOccurence( )  
nodejs function get_nextOccurence( )  
php function get_nextOccurence( )  
cpp s64 get_nextOccurence( )  
m -(s64) nextOccurence  
pas function get_nextOccurence( ): int64  
vb function get_nextOccurence( ) As Long  
cs long get_nextOccurence( )  
java long get_nextOccurence( )  
py def get_nextOccurence( )
```

**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).

js	function get(userData) {
node.js	function get(userData) {
php	function get(userData) {
cpp	void * get(userData) {
m	-(void*) userData
pas	function get(userData): Tobject
vb	function get(userData) As Object
cs	object get(userData)
java	Object get(userData)
py	def get(userData)

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()**  
**wakeupschedule→weekDays()**  
**wakeupschedule.get\_weekDays()****YWakeUpSchedule**

Retourne les jours de la semaine où le réveil est actif..

js	function get_weekDays( )
nodejs	function get_weekDays( )
php	function get_weekDays( )
cpp	int get_weekDays( )
m	-(int) weekDays
pas	function get_weekDays( ): LongInt
vb	function get_weekDays( ) As Integer
cs	int get_weekDays( )
java	int get_weekDays( )
py	def get_weekDays( )
cmd	YWakeUpSchedule target 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.

js	function <b>isOnline()</b>
node.js	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline()</b> : boolean
vb	function <b>isOnline()</b> As Boolean
cs	bool <b>isOnline()</b>
java	boolean <b>isOnline()</b>
py	def <b>isOnline()</b>

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()****YWakeUpSchedule**

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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.

<code>js</code>	<code>function load( msValidity)</code>
<code>node.js</code>	<code>function load( msValidity)</code>
<code>php</code>	<code>function load( \$msValidity)</code>
<code>cpp</code>	<code>YRETCODE load( int msValidity)</code>
<code>m</code>	<code>-(YRETCODE) load : (int) msValidity</code>
<code>pas</code>	<code>function load( msValidity: integer): YRETCODE</code>
<code>vb</code>	<code>function load( ByVal msValidity As Integer) As YRETCODE</code>
<code>cs</code>	<code>YRETCODE load( int msValidity)</code>
<code>java</code>	<code>int load( long msValidity)</code>
<code>py</code>	<code>def load( msValidity)</code>

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()

## YWakeUpSchedule

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()****YWakeUpSchedule**

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

<code>js</code>	<code>function nextWakeUpSchedule( )</code>
<code>node.js</code>	<code>function nextWakeUpSchedule( )</code>
<code>php</code>	<code>function nextWakeUpSchedule( )</code>
<code>cpp</code>	<code>YWakeUpSchedule * nextWakeUpSchedule( )</code>
<code>m</code>	<code>-(YWakeUpSchedule*) nextWakeUpSchedule</code>
<code>pas</code>	<code>function nextWakeUpSchedule( ): TYWakeUpSchedule</code>
<code>vb</code>	<code>function nextWakeUpSchedule( ) As YWakeUpSchedule</code>
<code>cs</code>	<code>YWakeUpSchedule nextWakeUpSchedule( )</code>
<code>java</code>	<code>YWakeUpSchedule nextWakeUpSchedule( )</code>
<code>py</code>	<code>def nextWakeUpSchedule( )</code>

**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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YWakeUpScheduleValueCallback callback)
m -(int) registerValueCallback : (YWakeUpScheduleValueCallback) callback
pas function registerValueCallback( callback: TYWakeUpScheduleValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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

js	function <b>set_hours( newval)</b>
node.js	function <b>set_hours( newval)</b>
php	function <b>set_hours( \$newval)</b>
cpp	int <b>set_hours( int newval)</b>
m	-(int) <b>setHours : (int) newval</b>
pas	function <b>set_hours( newval: LongInt): integer</b>
vb	function <b>set_hours( ByVal newval As Integer) As Integer</b>
cs	int <b>set_hours( int newval)</b>
java	int <b>set_hours( int newval)</b>
py	<b>def set_hours( newval)</b>
cmd	<b>YWakeUpSchedule target set_hours newval</b>

**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.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YWakeUpSchedule 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 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

<b>js</b>	function <b>set_minutes( bitmap)</b>
<b>node.js</b>	function <b>set_minutes( bitmap)</b>
<b>php</b>	function <b>set_minutes( \$bitmap)</b>
<b>cpp</b>	int <b>set_minutes( s64 bitmap)</b>
<b>m</b>	-(int) <b>setMinutes : (s64) bitmap</b>
<b>pas</b>	function <b>set_minutes( bitmap: int64): LongInt</b>
<b>vb</b>	function <b>set_minutes( ) As Integer</b>
<b>cs</b>	int <b>set_minutes( long bitmap)</b>
<b>java</b>	int <b>set_minutes( long bitmap)</b>
<b>py</b>	<b>def set_minutes( bitmap)</b>
<b>cmd</b>	<b>YWakeUpSchedule target set_minutes bitmap</b>

**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

```
js function set_minutesA( newval)
nodejs function set_minutesA( newval)
php function set_minutesA( $newval)
cpp int set_minutesA( int newval)
m -(int) setMinutesA : (int) newval
pas function set_minutesA( newval: LongInt): integer
vb function set_minutesA( ByVal newval As Integer) As Integer
cs int set_minutesA( int newval)
java int set_minutesA( int newval)
py def set_minutesA( newval)
cmd YWakeUpSchedule target 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.

js	function set_minutesB( newval)
node.js	function set_minutesB( newval)
php	function set_minutesB( \$newval)
cpp	int set_minutesB( int newval)
m	-(int) setMinutesB : (int) newval
pas	function set_minutesB( newval: LongInt): integer
vb	function set_minutesB( ByVal newval As Integer) As Integer
cs	int set_minutesB( int newval)
java	int set_minutesB( int newval)
py	def set_minutesB( newval)
cmd	YWakeUpSchedule target set_minutesB newval

**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

```
js function set_monthDays( newval)
nodejs function set_monthDays( newval)
php function set_monthDays( $newval)
cpp int set_monthDays( int newval)
m -(int) setMonthDays : (int) newval
pas function set_monthDays( newval: LongInt): integer
vb function set_monthDays( ByVal newval As Integer) As Integer
cs int set_monthDays( int newval)
java int set_monthDays( int newval)
py def set_monthDays( newval)
cmd YWakeUpSchedule target 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

js	function <b>set_months( newval)</b>
node.js	function <b>set_months( newval)</b>
php	function <b>set_months( \$newval)</b>
cpp	int <b>set_months( int newval)</b>
m	-(int) <b>setMonths : (int) newval</b>
pas	function <b>set_months( newval: LongInt): integer</b>
vb	function <b>set_months( ByVal newval As Integer) As Integer</b>
cs	int <b>set_months( int newval)</b>
java	int <b>set_months( int newval)</b>
py	<b>def set_months( newval)</b>
cmd	<b>YWakeUpSchedule target set_months newval</b>

**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).

```
js function set(userData) {  
nodejs function set(userData) {  
php function set(userData) {  
cpp void set(userData) {  
m -(void) setUserData : (void*) data  
pas procedure set(userData: Tobject)  
vb procedure set(userData( ByVal data As Object)  
cs void set(userData( object data  
java void set(userData( Object data  
py def set(userData( data)
```

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

```
js function set_weekDays( newval)
node.js function set_weekDays( newval)
php function set_weekDays( $newval)
cpp int set_weekDays( int newval)
m -(int) setWeekDays : (int) newval
pas function set_weekDays( newval: LongInt): integer
vb function set_weekDays( ByVal newval As Integer) As Integer
cs int set_weekDays( int newval)
java int set_weekDays( int newval)
py def set_weekDays( newval)
cmd YWakeUpSchedule target set_weekDays newval
```

**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()****YWakeUpSchedule**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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()****YWatchdog****yFindWatchdog() YWatchdog.FindWatchdog()**

Permet de retrouver un watchdog d'après un identifiant donné.

js	function <b>yFindWatchdog( func)</b>
node.js	function <b>FindWatchdog( func)</b>
php	function <b>yFindWatchdog( \$func)</b>
cpp	YWatchdog* <b>yFindWatchdog( const string&amp; func)</b>
m	YWatchdog* <b>yFindWatchdog( NSString* func)</b>
pas	function <b>yFindWatchdog( func: string): TYWatchdog</b>
vb	function <b>yFindWatchdog( ByVal func As String) As YWatchdog</b>
cs	YWatchdog <b>FindWatchdog( string func)</b>
java	YWatchdog <b>FindWatchdog( String func)</b>
py	def <b>FindWatchdog( func)</b>

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que 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() YWatchdog.FirstWatchdog()

## YWatchdog

Commence l'énumération des watchdog accessibles par la librairie.

js	function <b>yFirstWatchdog()</b>
nodejs	function <b>FirstWatchdog()</b>
php	function <b>yFirstWatchdog()</b>
cpp	YWatchdog* <b>yFirstWatchdog()</b>
m	YWatchdog* <b>yFirstWatchdog()</b>
pas	function <b>yFirstWatchdog()</b> : TYWatchdog
vb	function <b>yFirstWatchdog()</b> As YWatchdog
cs	YWatchdog <b>FirstWatchdog()</b>
java	YWatchdog <b>FirstWatchdog()</b>
py	def <b>FirstWatchdog()</b>

Utiliser la fonction `YWatchdog.nextWatchdog()` pour itérer sur les autres watchdog.

#### Retourne :

un pointeur sur un objet `YWatchdog`, correspondant à le 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

```
js function delayedPulse( ms_delay, ms_duration)
nodejs function delayedPulse( ms_delay, ms_duration)
php function delayedPulse( $ms_delay, $ms_duration)
cpp int delayedPulse( int ms_delay, int ms_duration)
m -(int) delayedPulse : (int) ms_delay : (int) ms_duration
pas function delayedPulse( ms_delay: LongInt, ms_duration: LongInt): integer
vb function delayedPulse( ByVal ms_delay As Integer,
                           ByVal ms_duration As Integer) As Integer
cs int delayedPulse( int ms_delay, int ms_duration)
java int delayedPulse( int ms_delay, int ms_duration)
py def delayedPulse( ms_delay, ms_duration)
cmd YWatchdog target delayedPulse ms_delay ms_duration
```

**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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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 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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YWatchdog target 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.

js	function <b>get_autoStart( )</b>
nodejs	function <b>get_autoStart( )</b>
php	function <b>get_autoStart( )</b>
cpp	Y_AUTOSTART_enum <b>get_autoStart( )</b>
m	-(Y_AUTOSTART_enum) <b>autoStart</b>
pas	function <b>get_autoStart( ): Integer</b>
vb	function <b>get_autoStart( ) As Integer</b>
cs	int <b>get_autoStart( )</b>
java	int <b>get_autoStart( )</b>
py	def <b>get_autoStart( )</b>
cmd	<b>YWatchdog target get_autoStart</b>

**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().

```
js function get_countdown( )  
nodejs function get_countdown( )  
php function get_countdown( )  
cpp s64 get_countdown( )  
m -(s64) countdown  
pas function get_countdown( ): int64  
vb function get_countdown( ) As Long  
cs long get_countdown( )  
java long get_countdown( )  
py def get_countdown( )  
cmd YWatchdog target 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.

js	function <b>get_errorMessage( )</b>
node.js	function <b>get_errorMessage( )</b>
php	function <b>get_errorMessage( )</b>
cpp	string <b>get_errorMessage( )</b>
m	-(NSString*) errorMessage
pas	function <b>get_errorMessage( )</b> : string
vb	function <b>get_errorMessage( )</b> As String
cs	string <b>get_errorMessage( )</b>
java	String <b>get_errorMessage( )</b>
py	def <b>get_errorMessage( )</b>

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

js	function get_friendlyName( )
nodejs	function get_friendlyName( )
php	function get_friendlyName( )
cpp	string get_friendlyName( )
m	-(NSString*) friendlyName
cs	string get_friendlyName( )
java	String get_friendlyName( )
py	def get_friendlyName( )

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId()</b>
node.js	function <b>get_functionId()</b>
php	function <b>get_functionId()</b>
cpp	string <b>get_functionId()</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId()</b> As String
cs	string <b>get_functionId()</b>
java	String <b>get_functionId()</b>
py	def <b>get_functionId()</b>

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.

js	function <b>get_hardwareId( )</b>
node.js	function <b>get_hardwareId( )</b>
php	function <b>get_hardwareId( )</b>
cpp	string <b>get_hardwareId( )</b>
m	-(NSString*) hardwareId
vb	function <b>get_hardwareId( ) As String</b>
cs	string <b>get_hardwareId( )</b>
java	String <b>get_hardwareId( )</b>
py	def <b>get_hardwareId( )</b>

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.

js	function <b>get_logicalName( )</b>
node.js	function <b>get_logicalName( )</b>
php	function <b>get_logicalName( )</b>
cpp	string <b>get_logicalName( )</b>
m	-(NSString*) logicalName
pas	function <b>get_logicalName( )</b> : string
vb	function <b>get_logicalName( )</b> As String
cs	string <b>get_logicalName( )</b>
java	String <b>get_logicalName( )</b>
py	<b>def get_logicalName( )</b>
cmd	YWatchdog <b>target get_logicalName</b>

**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.

```
js function get_maxTimeOnStateA( )
nodejs function get_maxTimeOnStateA( )
php function get_maxTimeOnStateA( )
cpp s64 get_maxTimeOnStateA( )
m -(s64) maxTimeOnStateA
pas function get_maxTimeOnStateA( ): int64
vb function get_maxTimeOnStateA( ) As Long
cs long get_maxTimeOnStateA( )
java long get_maxTimeOnStateA( )
py def get_maxTimeOnStateA( )
cmd YWatchdog target 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()**  
**watchdog→maxTimeOnStateB()**  
**watchdog.get\_maxTimeOnStateB()**

YWatchdog

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

js	function <b>get_maxTimeOnStateB()</b>
nodejs	function <b>get_maxTimeOnStateB()</b>
php	function <b>get_maxTimeOnStateB()</b>
cpp	s64 <b>get_maxTimeOnStateB()</b>
m	-(s64) maxTimeOnStateB
pas	function <b>get_maxTimeOnStateB()</b> : int64
vb	function <b>get_maxTimeOnStateB()</b> As Long
cs	long <b>get_maxTimeOnStateB()</b>
java	long <b>get_maxTimeOnStateB()</b>
py	def <b>get_maxTimeOnStateB()</b>
cmd	YWatchdog <b>target get_maxTimeOnStateB</b>

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.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<b>YModule * get_module()</b>
m	<b>-(YModule*) module</b>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As <b>YModule</b>
cs	<b>YModule get_module()</b>
java	<b>YModule get_module()</b>
py	<b>def get_module()</b>

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()

YWatchdog

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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.

```
js function get_output( )
node.js function get_output( )
php function get_output( )
cpp Y_OUTPUT_enum get_output( )
m -(Y_OUTPUT_enum) output
pas function get_output( ): Integer
vb function get_output( ) As Integer
cs int get_output( )
java int get_output( )
py def get_output( )
cmd YWatchdog target 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.

js	function <b>get_pulseTimer( )</b>
node.js	function <b>get_pulseTimer( )</b>
php	function <b>get_pulseTimer( )</b>
cpp	<b>s64 get_pulseTimer( )</b>
m	-(s64) pulseTimer
pas	function <b>get_pulseTimer( ): int64</b>
vb	function <b>get_pulseTimer( ) As Long</b>
cs	<b>long get_pulseTimer( )</b>
java	<b>long get_pulseTimer( )</b>
py	<b>def get_pulseTimer( )</b>
cmd	<b>YWatchdog target get_pulseTimer</b>

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\_PULSESETIMER\_INVALID**.

**watchdog→get\_running()****YWatchdog****watchdog→running()watchdog.get\_running()**

Retourne l'état du watchdog.

```
js function get_running( )  
node.js function get_running( )  
php function get_running( )  
cpp Y_RUNNING_enum get_running( )  
m -(Y_RUNNING_enum) running  
pas function get_running( ): Integer  
vb function get_running( ) As Integer  
cs int get_running( )  
java int get_running( )  
py def get_running( )  
cmd YWatchdog target get_running
```

**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).

js	function <b>get_state( )</b>
node.js	function <b>get_state( )</b>
php	function <b>get_state( )</b>
cpp	Y_STATE_enum <b>get_state( )</b>
m	-(Y_STATE_enum) state
pas	function <b>get_state( )</b> : Integer
vb	function <b>get_state( )</b> As Integer
cs	int <b>get_state( )</b>
java	int <b>get_state( )</b>
py	def <b>get_state( )</b>
cmd	YWatchdog <b>target get_state</b>

**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.

<b>watchdog→get_stateAtPowerOn()</b>	<b>YWatchdog</b>
<b>watchdog→stateAtPowerOn()</b>	
<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).

```
js function get_stateAtPowerOn( )
nodejs function get_stateAtPowerOn( )
php function get_stateAtPowerOn( )
cpp Y_STATEATPOWERON_enum get_stateAtPowerOn( )
m -(Y_STATEATPOWERON_enum) stateAtPowerOn
pas function get_stateAtPowerOn( ): Integer
vb function get_stateAtPowerOn( ) As Integer
cs int get_stateAtPowerOn( )
java int get_stateAtPowerOn( )
py def get_stateAtPowerOn( )
cmd YWatchdog target get_stateAtPowerOn
```

**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.

js	function <b>get_triggerDelay( )</b>
node.js	function <b>get_triggerDelay( )</b>
php	function <b>get_triggerDelay( )</b>
cpp	s64 <b>get_triggerDelay( )</b>
m	-(s64) triggerDelay
pas	function <b>get_triggerDelay( ): int64</b>
vb	function <b>get_triggerDelay( ) As Long</b>
cs	long <b>get_triggerDelay( )</b>
java	long <b>get_triggerDelay( )</b>
py	def <b>get_triggerDelay( )</b>
cmd	YWatchdog <b>target get_triggerDelay</b>

**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.

```
js function get_triggerDuration( )  
nodejs function get_triggerDuration( )  
php function get_triggerDuration( )  
cpp s64 get_triggerDuration( )  
m -(s64) triggerDuration  
pas function get_triggerDuration( ): int64  
vb function get_triggerDuration( ) As Long  
cs long get_triggerDuration( )  
java long get_triggerDuration( )  
py def get_triggerDuration( )  
cmd YWatchdog target 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).

js	function <b>get(userData)</b>
nodejs	function <b>get(userData)</b>
php	function <b>get(userData)</b>
cpp	void * <b>get(userData)</b>
m	-(void*) userData
pas	function <b>get(userData)</b> : Tobject
vb	function <b>get(userData)</b> As Object
cs	object <b>get(userData)</b>
java	Object <b>get(userData)</b>
py	<b>def get(userData)</b>

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.

```
js function isOnline( )
nodejs function isOnline( )
php function isOnline( )
cpp bool isOnline( )
m -(BOOL) isOnline
pas function isOnline( ): boolean
vb function isOnline( ) As Boolean
cs bool isOnline( )
java boolean isOnline( )
py def isOnline( )
```

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()

## YWatchdog

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context)
nodejs 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.

```
js function load( msValidity)
nodejs function load( msValidity)
php function load( $msValidity)
cpp YRETCODE load( int msValidity)
m -(YRETCODE) load : (int) msValidity
pas function load( msValidity: integer): YRETCODE
vb function load( ByVal msValidity As Integer) As YRETCODE
cs YRETCODE load( int msValidity)
java int load( long msValidity)
py def load( msValidity)
```

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()

## YWatchdog

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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()`.

js	function <b>nextWatchdog()</b>
node.js	function <b>nextWatchdog()</b>
php	function <b>nextWatchdog()</b>
cpp	YWatchdog * <b>nextWatchdog()</b>
m	-(YWatchdog*) <b>nextWatchdog</b>
pas	function <b>nextWatchdog()</b> : TYWatchdog
vb	function <b>nextWatchdog()</b> As YWatchdog
cs	YWatchdog <b>nextWatchdog()</b>
java	YWatchdog <b>nextWatchdog()</b>
py	def <b>nextWatchdog()</b>

**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).

```
js function pulse( ms_duration)
nodejs function pulse( ms_duration)
php function pulse( $ms_duration)
cpp int pulse( int ms_duration)
m -(int) pulse : (int) ms_duration
pas function pulse( ms_duration: LongInt): integer
vb function pulse( ByVal ms_duration As Integer) As Integer
cs int pulse( int ms_duration)
java int pulse( int ms_duration)
py def pulse( ms_duration)
cmd YWatchdog target pulse ms_duration
```

**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.

```
js function registerValueCallback( callback)
node.js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp int registerValueCallback( YWatchdogValueCallback callback)
m -(int) registerValueCallback : (YWatchdogValueCallback) callback
pas function registerValueCallback( callback: TYWatchdogValueCallback): LongInt
vb function registerValueCallback( ) As Integer
cs int registerValueCallback( ValueCallback callback)
java int registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas 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.

```
js function resetWatchdog( )
nodejs function resetWatchdog( )
php function resetWatchdog( )
cpp int resetWatchdog( )
m -(int) resetWatchdog
pas function resetWatchdog( ): integer
vb function resetWatchdog( ) As Integer
cs int resetWatchdog( )
java int resetWatchdog( )
py def resetWatchdog( )
cmd YWatchdog target 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.

js	function <b>set_autoStart( newval)</b>
node.js	function <b>set_autoStart( newval)</b>
php	function <b>set_autoStart( \$newval)</b>
cpp	int <b>set_autoStart( Y_AUTOSTART_enum newval)</b>
m	-(int) setAutoStart : (Y_AUTOSTART_enum) <b>newval</b>
pas	function <b>set_autoStart( newval: Integer): integer</b>
vb	function <b>set_autoStart( ByVal newval As Integer) As Integer</b>
cs	int <b>set_autoStart( int newval)</b>
java	int <b>set_autoStart( int newval)</b>
py	def <b>set_autoStart( newval)</b>
cmd	<b>YWatchdog target set_autoStart newval</b>

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.

js	function <b>set_logicalName( newval)</b>
node.js	function <b>set_logicalName( newval)</b>
php	function <b>set_logicalName( \$newval)</b>
cpp	int <b>set_logicalName( const string&amp; newval)</b>
m	-(int) <b>setLogicalName : (NSString*) newval</b>
pas	function <b>set_logicalName( newval: string): integer</b>
vb	function <b>set_logicalName( ByVal newval As String) As Integer</b>
cs	int <b>set_logicalName( string newval)</b>
java	int <b>set_logicalName( String newval)</b>
py	def <b>set_logicalName( newval)</b>
cmd	<b>YWatchdog target set_logicalName newval</b>

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique 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.

```
js function set_maxTimeOnStateA( newval)
nodejs function set_maxTimeOnStateA( newval)
php function set_maxTimeOnStateA( $newval)
cpp int set_maxTimeOnStateA( s64 newval)
m -(int) setMaxTimeOnStateA : (s64) newval
pas function set_maxTimeOnStateA( newval: int64): integer
vb function set_maxTimeOnStateA( ByVal newval As Long) As Integer
cs int set_maxTimeOnStateA( long newval)
java int set_maxTimeOnStateA( long newval)
py def set_maxTimeOnStateA( newval)
cmd YWatchdog target 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.

<b>js</b>	function <b>set_maxTimeOnStateB( newval)</b>
<b>node.js</b>	function <b>setMaxTimeOnStateB( newval)</b>
<b>php</b>	function <b>set_maxTimeOnStateB( \$newval)</b>
<b>cpp</b>	int <b>set_maxTimeOnStateB( s64 newval)</b>
<b>m</b>	-(int) setMaxTimeOnStateB : (s64) newval
<b>pas</b>	function <b>set_maxTimeOnStateB( newval: int64): integer</b>
<b>vb</b>	function <b>set_maxTimeOnStateB( ByVal newval As Long) As Integer</b>
<b>cs</b>	int <b>set_maxTimeOnStateB( long newval)</b>
<b>java</b>	int <b>set_maxTimeOnStateB( long newval)</b>
<b>py</b>	def <b>set_maxTimeOnStateB( newval)</b>
<b>cmd</b>	YWatchdog <b>target set_maxTimeOnStateB newval</b>

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.

js	function <b>set_output( newval)</b>
node.js	function <b>set_output( newval)</b>
php	function <b>set_output( \$newval)</b>
cpp	int <b>set_output( Y_OUTPUT_enum newval)</b>
m	-{int) setOutput : (Y_OUTPUT_enum) <b>newval</b>
pas	function <b>set_output( newval: Integer): integer</b>
vb	function <b>set_output( ByVal newval As Integer) As Integer</b>
cs	int <b>set_output( int newval)</b>
java	int <b>set_output( int newval)</b>
py	def <b>set_output( newval)</b>
cmd	<b>YWatchdog target set_output newval</b>

**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.

js	function <b>set_running( newval)</b>
nodejs	function <b>set_running( newval)</b>
php	function <b>set_running( \$newval)</b>
cpp	int <b>set_running( Y_RUNNING_enum newval)</b>
m	-(int) setRunning : (Y_RUNNING_enum) <b>newval</b>
pas	function <b>set_running( newval: Integer): integer</b>
vb	function <b>set_running( ByVal newval As Integer) As Integer</b>
cs	int <b>set_running( int newval)</b>
java	int <b>set_running( int newval)</b>
py	def <b>set_running( newval)</b>
cmd	<b>YWatchdog target set_running newval</b>

**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()**  
**watchdog→setState()watchdog.set\_state()****YWatchdog**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

js	function <b>set_state( newval)</b>
node.js	function <b>set_state( newval)</b>
php	function <b>set_state( \$newval)</b>
cpp	int <b>set_state( Y_STATE_enum newval)</b>
m	-{int) setState : (Y_STATE_enum) <b>newval</b>
pas	function <b>set_state( newval: Integer): integer</b>
vb	function <b>set_state( ByVal newval As Integer) As Integer</b>
cs	int <b>set_state( int newval)</b>
java	int <b>set_state( int newval)</b>
py	def <b>set_state( newval)</b>
cmd	<b>YWatchdog target set_state newval</b>

**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).

js	function <b>set_stateAtPowerOn( newval)</b>
nodejs	function <b>set_stateAtPowerOn( newval)</b>
php	function <b>set_stateAtPowerOn( \$newval)</b>
cpp	int <b>set_stateAtPowerOn( Y_STATEATPOWERON_enum newval)</b>
m	-(int) setStateAtPowerOn : (Y_STATEATPOWERON_enum) <b>newval</b>
pas	function <b>set_stateAtPowerOn( newval: Integer): integer</b>
vb	function <b>set_stateAtPowerOn( ByVal newval As Integer) As Integer</b>
cs	int <b>set_stateAtPowerOn( int newval)</b>
java	int <b>set_stateAtPowerOn( int newval)</b>
py	def <b>set_stateAtPowerOn( newval)</b>
cmd	YWatchdog <b>target set_stateAtPowerOn newval</b>

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.

js	function set_triggerDelay( newval)
nodejs	function set_triggerDelay( newval)
php	function set_triggerDelay( \$newval)
cpp	int set_triggerDelay( s64 newval)
m	-(int) setTriggerDelay : (s64) newval
pas	function set_triggerDelay( newval: int64): integer
vb	function set_triggerDelay( ByVal newval As Long) As Integer
cs	int set_triggerDelay( long newval)
java	int set_triggerDelay( long newval)
py	def set_triggerDelay( newval)
cmd	YWatchdog target 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.

<code>js</code>	<code>function set_triggerDuration( newval)</code>
<code>node.js</code>	<code>function set_triggerDuration( newval)</code>
<code>php</code>	<code>function set_triggerDuration( \$newval)</code>
<code>cpp</code>	<code>int set_triggerDuration( s64 newval)</code>
<code>m</code>	<code>-(int) setTriggerDuration : (s64) newval</code>
<code>pas</code>	<code>function set_triggerDuration( newval: int64): integer</code>
<code>vb</code>	<code>function set_triggerDuration( ByVal newval As Long) As Integer</code>
<code>cs</code>	<code>int set_triggerDuration( long newval)</code>
<code>java</code>	<code>int set_triggerDuration( long newval)</code>
<code>py</code>	<code>def set_triggerDuration( newval)</code>
<code>cmd</code>	<code>YWatchdog target set_triggerDuration newval</code>

**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).

```
js function set(userData) data
nodejs function set(userData) data
php function set(userData) $data
cpp void set(userData) void* data
m -(void) setUserData : (void*) data
pas procedure set(userData) data: Tobject
vb procedure set(userData) ByVal data As Object
cs void set(userData) object data
java void set(userData) Object data
py def set(userData) data
```

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()

## YWatchdog

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
nodejs 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()****YWireless****yFindWireless()YWireless.FindWireless()**

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

<b>js</b>	<code>function yFindWireless( func)</code>
<b>node.js</b>	<code>function FindWireless( func)</code>
<b>php</b>	<code>function yFindWireless( \$func)</code>
<b>cpp</b>	<code>YWireless* yFindWireless( string func)</code>
<b>m</b>	<code>+ (YWireless*) yFindWireless : (NSString*) func</code>
<b>pas</b>	<code>function yFindWireless( func: string): TYWireless</code>
<b>vb</b>	<code>function yFindWireless( ByVal func As String) As YWireless</code>
<b>cs</b>	<code>YWireless FindWireless( string func)</code>
<b>java</b>	<code>YWireless FindWireless( String func)</code>
<b>py</b>	<code>def FindWireless( func)</code>

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()****YWireless****yFirstWireless() YWireless.FirstWireless()**

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

js	function <b>yFirstWireless( )</b>
nodejs	function <b>FirstWireless( )</b>
php	function <b>yFirstWireless( )</b>
cpp	YWireless* <b>yFirstWireless( )</b>
m	YWireless* <b>yFirstWireless( )</b>
pas	function <b>yFirstWireless( ): TYWireless</b>
vb	function <b>yFirstWireless( ) As YWireless</b>
cs	YWireless <b>FirstWireless( )</b>
java	YWireless <b>FirstWireless( )</b>
py	def <b>FirstWireless( )</b>

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".

```

js   function adhocNetwork( ssid, securityKey)
nodejs function adhocNetwork( ssid, securityKey)
php  function adhocNetwork( $ssid, $securityKey)
cpp   int adhocNetwork( string ssid, string securityKey)
m    -(int) adhocNetwork : (NSString*) ssid
           : (NSString*) securityKey
pas   function adhocNetwork( ssid: string, securityKey: string): integer
vb    function adhocNetwork( ByVal ssid As String,
                           ByVal securityKey As String) As Integer
cs    int adhocNetwork( string ssid, string securityKey)
java  int adhocNetwork( String ssid, String securityKey)
py    def adhocNetwork( ssid, securityKey)
cmd   YWireless target 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.

js	function <b>describe</b> ( )
nodejs	function <b>describe</b> ( )
php	function <b>describe</b> ( )
cpp	string <b>describe</b> ( )
m	-(NSString*) <b>describe</b>
pas	function <b>describe</b> ( ): string
vb	function <b>describe</b> ( ) As String
cs	string <b>describe</b> ( )
java	String <b>describe</b> ( )
py	def <b>describe</b> ( )

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).

```
js function get_advertisedValue( )  
nodejs function get_advertisedValue( )  
php function get_advertisedValue( )  
cpp string get_advertisedValue( )  
m -(NSString*) advertisedValue  
pas function get_advertisedValue( ): string  
vb function get_advertisedValue( ) As String  
cs string get_advertisedValue( )  
java String get_advertisedValue( )  
py def get_advertisedValue( )  
cmd YWireless target 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.

js	function <b>get_channel( )</b>
nodejs	function <b>get_channel( )</b>
php	function <b>get_channel( )</b>
cpp	int <b>get_channel( )</b>
m	-(int) channel
pas	function <b>get_channel( ): LongInt</b>
vb	function <b>get_channel( ) As Integer</b>
cs	int <b>get_channel( )</b>
java	int <b>get_channel( )</b>
py	def <b>get_channel( )</b>
cmd	<b>YWireless target get_channel</b>

**Retourne :**

un entier représentant le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé

En cas d'erreur, déclenche une exception ou retourne **Y\_CHANNEL\_INVALID**.

**wireless→get\_detectedWlans()**  
**wireless→detectedWlans()**  
**wireless.get\_detectedWlans()**

**YWireless**

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

```
js function get_detectedWlans( )  
nodejs function get_detectedWlans( )  
php function get_detectedWlans( )  
cpp vector<YWlanRecord> get_detectedWlans( )  
m -(NSMutableArray*) detectedWlans  
pas function get_detectedWlans( ): TYWlanRecordArray  
vb function get_detectedWlans( ) As List  
cs List<YWlanRecord> get_detectedWlans( )  
java ArrayList<YWlanRecord> get_detectedWlans( )  
py def get_detectedWlans( )  
cmd YWireless target 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.

```
js function get_errorMessage( )
nodejs function get_errorMessage( )
php function get_errorMessage( )
cpp string get_errorMessage( )
m -(NSString*) errorMessage
pas function get_errorMessage( ): string
vb function get_errorMessage( ) As String
cs string get_errorMessage( )
java String get_errorMessage( )
py def get_errorMessage( )
```

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.

```
js function get_errorType( )  
nodejs function get_errorType( )  
php function get_errorType( )  
cpp YRETCODE get_errorType( )  
pas function get_errorType( ): YRETCODE  
vb function get_errorType( ) As YRETCODE  
cs YRETCODE get_errorType( )  
java int get_errorType( )  
py def get_errorType( )
```

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.

```
js function get_friendlyName( )
nodejs function get_friendlyName( )
php function get_friendlyName( )
cpp string get_friendlyName( )
m -(NSString*) friendlyName
cs string get_friendlyName( )
java String get_friendlyName( )
py def get_friendlyName( )
```

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.

js	function get_functionDescriptor( )
node.js	function get_functionDescriptor( )
php	function get_functionDescriptor( )
cpp	YFUN_DESCR get_functionDescriptor( )
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor( ): YFUN_DESCR
vb	function get_functionDescriptor( ) As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor( )
java	String get_functionDescriptor( )
py	def get_functionDescriptor( )

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.

js	function <b>get_functionId( )</b>
node.js	function <b>get_functionId( )</b>
php	function <b>get_functionId( )</b>
cpp	string <b>get_functionId( )</b>
m	-(NSString*) <b>functionId</b>
vb	function <b>get_functionId( ) As String</b>
cs	string <b>get_functionId( )</b>
java	String <b>get_functionId( )</b>
py	def <b>get_functionId( )</b>

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→hardwareId()wireless.get\_hwrid()**

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

js	function <b>get_hwrid()</b> {
nodejs	function <b>get_hwrid()</b> {
php	function <b>get_hwrid()</b> {
cpp	string <b>get_hwrid()</b> {
m	- NSString* hardwareId
vb	function <b>get_hwrid()</b> As String
cs	string <b>get_hwrid()</b> {
java	String <b>get_hwrid()</b> {
py	def <b>get_hwrid()</b> {

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.

```
js function get_linkQuality( )
nodejs function get_linkQuality( )
php function get_linkQuality( )
cpp int get_linkQuality( )
m -(int) linkQuality
pas function get_linkQuality( ): LongInt
vb function get_linkQuality( ) As Integer
cs int get_linkQuality( )
java int get_linkQuality( )
py def get_linkQuality( )
cmd YWireless target get_linkQuality
```

**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.

```
js function get_logicalName( )  
node.js function get_logicalName( )  
php function get_logicalName( )  
cpp string get_logicalName( )  
m -(NSString*) logicalName  
pas function get_logicalName( ): string  
vb function get_logicalName( ) As String  
cs string get_logicalName( )  
java String get_logicalName( )  
py def get_logicalName( )  
cmd YWireless target 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.

js	function <b>get_message( )</b>
node.js	function <b>get_message( )</b>
php	function <b>get_message( )</b>
cpp	string <b>get_message( )</b>
m	-(NSString*) <b>message</b>
pas	function <b>get_message( ): string</b>
vb	function <b>get_message( ) As String</b>
cs	string <b>get_message( )</b>
java	String <b>get_message( )</b>
py	def <b>get_message( )</b>
cmd	<b>YWIRELESS target get_message</b>

**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.

js	function <b>get_module()</b>
node.js	function <b>get_module()</b>
php	function <b>get_module()</b>
cpp	<code>YModule * get_module()</code>
m	<code>-(YModule*) module</code>
pas	function <b>get_module()</b> : TYModule
vb	function <b>get_module()</b> As YModule
cs	<code>YModule get_module()</code>
java	<code>YModule get_module()</code>
py	<code>def get_module()</code>

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()

YWireless

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
js   function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornnée 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.

```
js function get_security( )
node.js function get_security( )
php function get_security( )
cpp Y_SECURITY_enum get_security( )
m -(Y_SECURITY_enum) security
pas function get_security( ): Integer
vb function get_security( ) As Integer
cs int get_security( )
java int get_security( )
py def get_security( )
cmd YWireless target 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.

js	function <b>get_ssid()</b>
node.js	function <b>get_ssid()</b>
php	function <b>get_ssid()</b>
cpp	string <b>get_ssid()</b>
m	-(NSString*) ssid
pas	function <b>get_ssid()</b> : string
vb	function <b>get_ssid()</b> As String
cs	string <b>get_ssid()</b>
java	String <b>get_ssid()</b>
py	def <b>get_ssid()</b>
cmd	YWireless target <b>get_ssid</b>

**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).

```
js function get(userData) {  
nodejs function get(userData) {  
php function get(userData) {  
cpp void * get(userData)  
m -(void*) userData  
pas function get(userData): Object  
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.

## wireless→isOnline()wireless.isOnline()

## YWireless

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

js	function <b>isOnline</b> ( )
node.js	function <b>isOnline</b> ( )
php	function <b>isOnline</b> ( )
cpp	bool <b>isOnline</b> ( )
m	-(BOOL) <b>isOnline</b>
pas	function <b>isOnline</b> ( ): boolean
vb	function <b>isOnline</b> ( ) As Boolean
cs	bool <b>isOnline</b> ( )
java	boolean <b>isOnline</b> ( )
py	def <b>isOnline</b> ( )

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()

YWireless

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

```
js   function isOnline_async( callback, context )
nodejs 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").

```

js   function joinNetwork( ssid, securityKey)
nodejs function joinNetwork( ssid, securityKey)
php  function joinNetwork( $ssid, $securityKey)
cpp   int joinNetwork( string ssid, string securityKey)
m    -(int) joinNetwork : (NSString*) ssid
           : (NSString*) securityKey
pas   function joinNetwork( ssid: string, securityKey: string): integer
vb    function joinNetwork( ByVal ssid As String,
                           ByVal securityKey As String) As Integer
cs    int joinNetwork( string ssid, string securityKey)
java  int joinNetwork( String ssid, String securityKey)
py    def joinNetwork( ssid, securityKey)
cmd   YWireless target joinNetwork ssid securityKey

```

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.

```
js   function load( msValidity)
nodejs function load( msValidity)
php  function load( $msValidity)
cpp   YRETCODE load( int msValidity)
m    -(YRETCODE) load : (int) msValidity
pas   function load( msValidity: integer): YRETCODE
vb    function load( ByVal msValidity As Integer) As YRETCODE
cs    YRETCODE load( int msValidity)
java  int load( long msValidity)
py    def load( msValidity)
```

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()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
js   function load_async( msValidity, callback, context)
nodejs 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()`.

js	<b>function nextWireless( )</b>
nodejs	<b>function nextWireless( )</b>
php	<b>function nextWireless( )</b>
cpp	<b>YWireless * nextWireless( )</b>
m	<b>-(YWireless*) nextWireless</b>
pas	<b>function nextWireless( ): TYWireless</b>
vb	<b>function nextWireless( ) As YWireless</b>
cs	<b>YWireless nextWireless( )</b>
java	<b>YWireless nextWireless( )</b>
py	<b>def nextWireless( )</b>

**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.

js	function <b>registerValueCallback</b> ( <b>callback</b> )
node.js	function <b>registerValueCallback</b> ( <b>callback</b> )
php	function <b>registerValueCallback</b> ( <b>\$callback</b> )
cpp	int <b>registerValueCallback</b> ( YWirelessValueCallback <b>callback</b> )
m	-(int) <b>registerValueCallback</b> : (YWirelessValueCallback) <b>callback</b>
pas	function <b>registerValueCallback</b> ( <b>callback</b> : TYWirelessValueCallback): LongInt
vb	function <b>registerValueCallback</b> ( ) As Integer
cs	int <b>registerValueCallback</b> ( ValueCallback <b>callback</b> )
java	int <b>registerValueCallback</b> ( UpdateCallback <b>callback</b> )
py	def <b>registerValueCallback</b> ( <b>callback</b> )

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

### Paramètres :

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wireless→set\_logicalName()  
wireless→setLogicalName()  
wireless.set\_logicalName()

YWireless

Modifie le nom logique de l'interface réseau sans fil.

```
js function set_logicalName( newval)
nodejs function set_logicalName( newval)
php function set_logicalName( $newval)
cpp int set_logicalName( const string& newval)
m -(int) setLogicalName : (NSString*) newval
pas function set_logicalName( newval: string): integer
vb function set_logicalName( ByVal newval As String) As Integer
cs int set_logicalName( string newval)
java int set_logicalName( String newval)
py def set_logicalName( newval)
cmd YWireless 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 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).

js	function set(userData( data)
node.js	function set(userData( data)
php	function set(userData( \$data)
cpp	void set(userData( void* data)
m	-(void) setUserData : (void*) data
pas	procedure set(userData( data: Tobject)
vb	procedure set(userData( ByVal data As Object)
cs	void set(userData( object data)
java	void set(userData( Object data)
py	def set(userData( data)

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()****YWireless**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js function wait_async( callback, context)
node.js 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 39  
adhocNetwork, YWireless 1731  
Alimentation 494  
AnButton 85

## B

Blueprint 12  
Brute 348

## C

C# 3  
calibrate, YLightSensor 768  
calibrateFromPoints, YAccelerometer 43  
calibrateFromPoints, YCarbonDioxide 131  
calibrateFromPoints, YCompass 207  
calibrateFromPoints, YCurrent 251  
calibrateFromPoints, YGenericSensor 560  
calibrateFromPoints, YGyro 610  
calibrateFromPoints, YHumidity 694  
calibrateFromPoints, YLightSensor 769  
calibrateFromPoints, YMagnetometer 812  
calibrateFromPoints, YPower 998  
calibrateFromPoints, YPressure 1045  
calibrateFromPoints, YQt 1157  
calibrateFromPoints,YSensor 1311  
calibrateFromPoints, YTemperature 1393  
calibrateFromPoints, YTilt 1438  
calibrateFromPoints, YVoc 1481  
calibrateFromPoints, YVoltage 1524  
callbackLogin, YNetwork 911  
cancel3DCalibration, YRefFrame 1231  
CarbonDioxide 127  
CheckLogicalName, YAPI 14  
clear, YDisplayLayer 463  
clearConsole, YDisplayLayer 464  
ColorLed 170  
Compass 203  
Configuration 1227  
consoleOut, YDisplayLayer 465  
Contrôle 4, 6, 494, 860, 967  
copyLayerContent, YDisplay 415  
Current 247

## D

DataLogger 290  
delayedPulse, YDigitalIO 367  
delayedPulse, YRelay 1271  
delayedPulse, YWatchdog 1683  
describe, YAccelerometer 44  
describe, YAnButton 89  
describe, YCarbonDioxide 132

describe, YColorLed 173  
describe, YCompass 208  
describe, YCurrent 252  
describe, YDataLogger 293  
describe, YDigitalIO 368  
describe, YDisplay 416  
describe, YDualPower 497  
describe, YFiles 526  
describe, YGenericSensor 561  
describe, YGyro 611  
describe, YHubPort 664  
describe, YHumidity 695  
describe, YLed 736  
describe, YLightSensor 770  
describe, YMagnetometer 813  
describe, YModule 864  
describe, YNetwork 912  
describe, YOsControl 970  
describe, YPower 999  
describe, YPressure 1046  
describe, YPwmOutput 1088  
describe, YPwmPowerSource 1129  
describe, YQt 1158  
describe, YRealTimeClock 1199  
describe, YRefFrame 1232  
describe, YRelay 1272  
describe, YSensor 1312  
describe, YServo 1354  
describe, YTemperature 1394  
describe, YTilt 1439  
describe, YVoc 1482  
describe, YVoltage 1525  
describe, YVSource 1566  
describe, YWakeUpMonitor 1603  
describe, YWakeUpSchedule 1642  
describe, YWatchdog 1684  
describe, YWireless 1732  
DigitalIO 363  
DisableExceptions, YAPI 15  
Display 411  
DisplayLayer 462  
Données 325, 335, 348  
download, YFiles 527  
download, YModule 865  
download\_async, YFiles 528  
drawBar, YDisplayLayer 466  
drawBitmap, YDisplayLayer 467  
drawCircle, YDisplayLayer 468  
drawDisc, YDisplayLayer 469  
drawImage, YDisplayLayer 470  
drawPixel, YDisplayLayer 471  
drawRect, YDisplayLayer 472  
drawText, YDisplayLayer 473  
dutyCycleMove, YPwmOutput 1089

## E

EnableExceptions, YAPI 16  
EnableUSBHost, YAPI 17  
Enregistrées 335, 348  
Erreurs 8

## F

fade, YDisplay 417  
Files 523  
FindAccelerometer, YAccelerometer 41  
FindAnButton, YAnButton 87  
FindCarbonDioxide, YCarbonDioxide 129  
FindColorLed, YColorLed 171  
FindCompass, YCompass 205  
FindCurrent, YCurrent 249  
FindDataLogger, YDataLogger 291  
FindDigitalIO, YDigitalIO 365  
FindDisplay, YDisplay 413  
FindDualPower, YDualPower 495  
FindFiles, YFiles 524  
FindGenericSensor, YGenericSensor 558  
FindGyro, YGyro 608  
FindHubPort, YHubPort 662  
FindHumidity, YHumidity 692  
FindLed, YLed 734  
FindLightSensor, YLightSensor 766  
FindMagnetometer, YMagnetometer 810  
FindModule, YModule 862  
FindNetwork, YNetwork 909  
FindOsControl, YOsControl 968  
FindPower, YPower 996  
FindPressure, YPressure 1043  
FindPwmOutput, YPwmOutput 1086  
FindPwmPowerSource, YPwmPowerSource 1127  
FindQt, YQt 1155  
FindRealTimeClock, YRealTimeClock 1197  
FindRefFrame, YRefFrame 1229  
FindRelay, YRelay 1269  
FindSensor, YSensor 1309  
FindServo,YServo 1352  
FindTemperature, YTemperature 1391  
FindTilt, YTilt 1436  
FindVoc, YVoc 1479  
FindVoltage, YVoltage 1522  
FindVSource, YVSource 1564  
FindWakeUpMonitor, YWakeUpMonitor 1601  
FindWakeUpSchedule, YWakeUpSchedule 1640  
FindWatchdog, YWatchdog 1681  
FindWireless, YWireless 1729  
FirstAccelerometer, YAccelerometer 42  
FirstAnButton, YAnButton 88  
FirstCarbonDioxide, YCarbonDioxide 130  
FirstColorLed, YColorLed 172  
FirstCompass, YCompass 206  
FirstCurrent, YCurrent 250  
FirstDataLogger, YDataLogger 292  
FirstDigitalIO, YDigitalIO 366

FirstDisplay, YDisplay 414  
FirstDualPower, YDualPower 496  
FirstFiles, YFiles 525  
FirstGenericSensor, YGenericSensor 559  
FirstGyro, YGyro 609  
FirstHubPort, YHubPort 663  
FirstHumidity, YHumidity 693  
FirstLed, YLed 735  
FirstLightSensor, YLightSensor 767  
FirstMagnetometer, YMagnetometer 811  
FirstModule, YModule 863  
FirstNetwork, YNetwork 910  
FirstOsControl, YOsControl 969  
FirstPower, YPower 997  
FirstPressure, YPressure 1044  
FirstPwmOutput, YPwmOutput 1087  
FirstPwmPowerSource, YPwmPowerSource 1128  
FirstQt, YQt 1156  
FirstRealTimeClock, YRealTimeClock 1198  
FirstRefFrame, YRefFrame 1230  
FirstRelay, YRelay 1270  
FirstSensor, YSensor 1310  
FirstServo, YServo 1353  
FirstTemperature, YTemperature 1392  
FirstTilt, YTilt 1437  
FirstVoc, YVoc 1480  
FirstVoltage, YVoltage 1523  
FirstVSource, YVSource 1565  
FirstWakeUpMonitor, YWakeUpMonitor 1602  
FirstWakeUpSchedule, YWakeUpSchedule 1641  
FirstWatchdog, YWatchdog 1682  
FirstWireless, YWireless 1730  
Fonctions 13, 1307  
forgetAllDataStreams, YDataLogger 294  
format\_fs, YFiles 529  
Forme 325  
FreeAPI, YAPI 18  
functionCount, YModule 866  
functionId, YModule 867  
functionName, YModule 868  
functionValue, YModule 869

## G

GenericSensor 556  
get\_3DCalibrationHint, YRefFrame 1233  
get\_3DCalibrationLogMsg, YRefFrame 1234  
get\_3DCalibrationProgress, YRefFrame 1235  
get\_3DCalibrationStage, YRefFrame 1236  
get\_3DCalibrationStageProgress, YRefFrame 1237  
get\_adminPassword, YNetwork 913  
get\_advertisedValue, YAccelerometer 45  
get\_advertisedValue, YAnButton 90  
get\_advertisedValue, YCarbonDioxide 133  
get\_advertisedValue, YColorLed 174  
get\_advertisedValue, YCompass 209  
get\_advertisedValue, YCurrent 253  
get\_advertisedValue, YDataLogger 295

get\_advertisedValue, YDigitalIO 369  
get\_advertisedValue, YDisplay 418  
get\_advertisedValue, YDualPower 498  
get\_advertisedValue, YFiles 530  
get\_advertisedValue, YGenericSensor 562  
get\_advertisedValue, YGyro 612  
get\_advertisedValue, YHubPort 665  
get\_advertisedValue, YHumidity 696  
get\_advertisedValue, YLed 737  
get\_advertisedValue, YLightSensor 771  
get\_advertisedValue, YMagnetometer 814  
get\_advertisedValue, YNetwork 914  
get\_advertisedValue, YOsControl 971  
get\_advertisedValue, YPower 1000  
get\_advertisedValue, YPressure 1047  
get\_advertisedValue, YPwmOutput 1090  
get\_advertisedValue, YPwmPowerSource 1130  
get\_advertisedValue, YQt 1159  
get\_advertisedValue, YRealTimeClock 1200  
get\_advertisedValue, YRefFrame 1238  
get\_advertisedValue, YRelay 1273  
get\_advertisedValue,YSensor 1313  
get\_advertisedValue,YServo 1355  
get\_advertisedValue,YTemperature 1395  
get\_advertisedValue, YTilt 1440  
get\_advertisedValue, YVoc 1483  
get\_advertisedValue, YVoltage 1526  
get\_advertisedValue, YVSource 1567  
get\_advertisedValue, YWakeUpMonitor 1604  
get\_advertisedValue, YWakeUpSchedule 1643  
get\_advertisedValue, YWatchdog 1685  
get\_advertisedValue, YWireless 1733  
get\_analogCalibration, YAnButton 91  
get\_autoStart, YDataLogger 296  
get\_autoStart, YWatchdog 1686  
get\_averageValue, YDataRun 325  
get\_averageValue, YDataStream 349  
get\_averageValue, YMeasure 854  
get\_baudRate, YHubPort 666  
get\_beacon, YModule 870  
get\_bearing, YRefFrame 1239  
get\_bitDirection, YDigitalIO 370  
get\_bitOpenDrain, YDigitalIO 371  
get\_bitPolarity, YDigitalIO 372  
get\_bitState, YDigitalIO 373  
get\_blinking, YLed 738  
get\_brightness, YDisplay 419  
get\_calibratedValue, YAnButton 92  
get\_calibrationMax, YAnButton 93  
get\_calibrationMin, YAnButton 94  
get\_callbackCredentials, YNetwork 915  
get\_callbackEncoding, YNetwork 916  
get\_callbackMaxDelay, YNetwork 917  
get\_callbackMethod, YNetwork 918  
get\_callbackMinDelay, YNetwork 919  
get\_callbackUrl, YNetwork 920  
get\_channel, YWireless 1734  
get\_columnCount, YDataStream 350  
get\_columnNames, YDataStream 351  
get\_cosPhi, YPower 1001  
get\_countdown, YRelay 1274  
get\_countdown, YWatchdog 1687  
get\_currentRawValue, YAccelerometer 46  
get\_currentRawValue, YCarbonDioxide 134  
get\_currentRawValue, YCompass 210  
get\_currentRawValue, YCurrent 254  
get\_currentRawValue, YGenericSensor 563  
get\_currentRawValue, YGyro 613  
get\_currentRawValue, YHumidity 697  
get\_currentRawValue, YLightSensor 772  
get\_currentRawValue, YMagnetometer 815  
get\_currentRawValue, YPower 1002  
get\_currentRawValue, YPressure 1048  
get\_currentRawValue, YQt 1160  
get\_currentRawValue, YSensor 1314  
get\_currentRawValue, YTemperature 1396  
get\_currentRawValue, YTilt 1441  
get\_currentRawValue, YVoc 1484  
get\_currentRawValue, YVoltage 1527  
get\_currentRunIndex, YDataLogger 297  
get\_currentValue, YAccelerometer 47  
get\_currentValue, YCarbonDioxide 135  
get\_currentValue, YCompass 211  
get\_currentValue, YCurrent 255  
get\_currentValue, YGenericSensor 564  
get\_currentValue, YGyro 614  
get\_currentValue, YHumidity 698  
get\_currentValue, YLightSensor 773  
get\_currentValue, YMagnetometer 816  
get\_currentValue, YPower 1003  
get\_currentValue, YPressure 1049  
get\_currentValue, YQt 1161  
get\_currentValue, YSensor 1315  
get\_currentValue, YTemperature 1397  
get\_currentValue, YTilt 1442  
get\_currentValue, YVoc 1485  
get\_currentValue, YVoltage 1528  
get\_data, YDataStream 352  
get\_dataRows, YDataStream 353  
get\_dataSamplesIntervalMs, YDataStream 354  
get\_dataSets, YDataLogger 298  
get\_dataStreams, YDataLogger 299  
get\_dateTime, YRealTimeClock 1201  
get\_detectedWlans, YWireless 1735  
get\_discoverable, YNetwork 921  
get\_display, YDisplayLayer 474  
get\_displayHeight, YDisplay 420  
get\_displayHeight, YDisplayLayer 475  
get\_displayLayer, YDisplay 421  
get\_displayType, YDisplay 422  
get\_displayWidth, YDisplay 423  
get\_displayWidth, YDisplayLayer 476  
get\_duration, YDataRun 326  
get\_duration, YDataStream 355  
get\_dutyCycle, YPwmOutput 1091  
get\_dutyCycleAtPowerOn, YPwmOutput 1092  
get\_enabled, YDisplay 424  
get\_enabled, YHubPort 667

get\_enabled, YPwmOutput 1093  
get\_enabled,YServo 1356  
get\_enabledAtPowerOn, YPwmOutput 1094  
get\_enabledAtPowerOn,YServo 1357  
get\_endTimeUTC, YDataSet 336  
get\_endTimeUTC,YMeasure 855  
get\_errorMessage, YAccelerometer 48  
get\_errorMessage, YAnButton 95  
get\_errorMessage, YCarbonDioxide 136  
get\_errorMessage, YColorLed 175  
get\_errorMessage, YCompass 212  
get\_errorMessage, YCurrent 256  
get\_errorMessage, YDataLogger 300  
get\_errorMessage, YDigitalIO 374  
get\_errorMessage, YDisplay 425  
get\_errorMessage, YDualPower 499  
get\_errorMessage, YFiles 531  
get\_errorMessage, YGenericSensor 565  
get\_errorMessage, YGyro 615  
get\_errorMessage, YHubPort 668  
get\_errorMessage, YHumidity 699  
get\_errorMessage, YLed 739  
get\_errorMessage, YLightSensor 774  
get\_errorMessage, YMagnetometer 817  
get\_errorMessage, YModule 871  
get\_errorMessage, YNetwork 922  
get\_errorMessage, YOsControl 972  
get\_errorMessage, YPower 1004  
get\_errorMessage, YPressure 1050  
get\_errorMessage, YPwmOutput 1095  
get\_errorMessage, YPwmPowerSource 1131  
get\_errorMessage, YQt 1162  
get\_errorMessage, YRealTimeClock 1202  
get\_errorMessage, YRefFrame 1240  
get\_errorMessage, YRelay 1275  
get\_errorMessage, YSensor 1316  
get\_errorMessage, YServo 1358  
get\_errorMessage, YTemperature 1398  
get\_errorMessage, YTilt 1443  
get\_errorMessage, YVoc 1486  
get\_errorMessage, YVoltage 1529  
get\_errorMessage, YVSource 1568  
get\_errorMessage, YWakeUpMonitor 1605  
get\_errorMessage, YWakeUpSchedule 1644  
get\_errorMessage, YWatchdog 1688  
get\_errorMessage, YWireless 1736  
get\_errorType, YAccelerometer 49  
get\_errorType, YAnButton 96  
get\_errorType, YCarbonDioxide 137  
get\_errorType, YColorLed 176  
get\_errorType, YCompass 213  
get\_errorType, YCurrent 257  
get\_errorType, YDataLogger 301  
get\_errorType, YDigitalIO 375  
get\_errorType, YDisplay 426  
get\_errorType, YDualPower 500  
get\_errorType, YFiles 532  
get\_errorType, YGenericSensor 566  
get\_errorType, YGyro 616  
get\_errorType, YHubPort 669  
get\_errorType, YHumidity 700  
get\_errorType, YLed 740  
get\_errorType, YLightSensor 775  
get\_errorType, YMagnetometer 818  
get\_errorType, YModule 872  
get\_errorType, YNetwork 923  
get\_errorType, YOsControl 973  
get\_errorType, YPower 1005  
get\_errorType, YPressure 1051  
get\_errorType, YPwmOutput 1096  
get\_errorType, YPwmPowerSource 1132  
get\_errorType, YQt 1163  
get\_errorType, YRealTimeClock 1203  
get\_errorType, YRefFrame 1241  
get\_errorType, YRelay 1276  
get\_errorType, YSensor 1317  
get\_errorType, YServo 1359  
get\_errorType, YTemperature 1399  
get\_errorType, YTilt 1444  
get\_errorType, YVoc 1487  
get\_errorType, YVoltage 1530  
get\_errorType, YVSource 1569  
get\_errorType, YWakeUpMonitor 1606  
get\_errorType, YWakeUpSchedule 1645  
get\_errorType, YWatchdog 1689  
get\_errorType, YWireless 1737  
get\_extPowerFailure, YVSource 1570  
get\_extVoltage, YDualPower 501  
get\_failure, YVSource 1571  
get\_filesCount, YFiles 533  
get\_firmwareRelease, YModule 873  
get\_freeSpace, YFiles 534  
get\_frequency, YPwmOutput 1097  
get\_friendlyName, YAccelerometer 50  
get\_friendlyName, YAnButton 97  
get\_friendlyName, YCarbonDioxide 138  
get\_friendlyName, YColorLed 177  
get\_friendlyName, YCompass 214  
get\_friendlyName, YCurrent 258  
get\_friendlyName, YDataLogger 302  
get\_friendlyName, YDigitalIO 376  
get\_friendlyName, YDisplay 427  
get\_friendlyName, YDualPower 502  
get\_friendlyName, YFiles 535  
get\_friendlyName, YGenericSensor 567  
get\_friendlyName, YGyro 617  
get\_friendlyName, YHubPort 670  
get\_friendlyName, YHumidity 701  
get\_friendlyName, YLed 741  
get\_friendlyName, YLightSensor 776  
get\_friendlyName, YMagnetometer 819  
get\_friendlyName, YNetwork 924  
get\_friendlyName, YOsControl 974  
get\_friendlyName, YPower 1006  
get\_friendlyName, YPressure 1052  
get\_friendlyName, YPwmOutput 1098  
get\_friendlyName, YPwmPowerSource 1133  
get\_friendlyName, YQt 1164

get\_friendlyName, YRealTimeClock 1204  
get\_friendlyName, YRefFrame 1242  
get\_friendlyName, YRelay 1277  
get\_friendlyName,YSensor 1318  
get\_friendlyName,YServo 1360  
get\_friendlyName,YTemperature 1400  
get\_friendlyName,YTilt 1445  
get\_friendlyName,YVoc 1488  
get\_friendlyName,YVoltage 1531  
get\_friendlyName,YVSource 1572  
get\_friendlyName,YWakeUpMonitor 1607  
get\_friendlyName,YWakeUpSchedule 1646  
get\_friendlyName,YWatchdog 1690  
get\_friendlyName,YWireless 1738  
get\_functionDescriptor, YAccelerometer 51  
get\_functionDescriptor, YAnButton 98  
get\_functionDescriptor, YCarbonDioxide 139  
get\_functionDescriptor, YColorLed 178  
get\_functionDescriptor, YCompass 215  
get\_functionDescriptor, YCurrent 259  
get\_functionDescriptor, YDataLogger 303  
get\_functionDescriptor, YDigitalIO 377  
get\_functionDescriptor, YDisplay 428  
get\_functionDescriptor, YDualPower 503  
get\_functionDescriptor, YFiles 536  
get\_functionDescriptor, YGenericSensor 568  
get\_functionDescriptor, YGyro 618  
get\_functionDescriptor, YHubPort 671  
get\_functionDescriptor, YHumidity 702  
get\_functionDescriptor, YLed 742  
get\_functionDescriptor, YLightSensor 777  
get\_functionDescriptor, YMagnetometer 820  
get\_functionDescriptor, YNetwork 925  
get\_functionDescriptor, YOsControl 975  
get\_functionDescriptor, YPower 1007  
get\_functionDescriptor, YPressure 1053  
get\_functionDescriptor, YPwmOutput 1099  
get\_functionDescriptor, YPwmPowerSource 1134  
get\_functionDescriptor, YQt 1165  
get\_functionDescriptor, YRealTimeClock 1205  
get\_functionDescriptor, YRefFrame 1243  
get\_functionDescriptor, YRelay 1278  
get\_functionDescriptor,YSensor 1319  
get\_functionDescriptor,YServo 1361  
get\_functionDescriptor,YTemperature 1401  
get\_functionDescriptor,YTilt 1446  
get\_functionDescriptor,YVoc 1489  
get\_functionDescriptor,YVoltage 1532  
get\_functionDescriptor,YVSource 1573  
get\_functionDescriptor,YWakeUpMonitor 1608  
get\_functionDescriptor,YWakeUpSchedule 1647  
get\_functionDescriptor,YWatchdog 1691  
get\_functionDescriptor,YWireless 1739  
get\_functionDescriptor, YAccelerometer 52  
get\_functionDescriptor, YAnButton 99  
get\_functionDescriptor, YCarbonDioxide 140  
get\_functionDescriptor, YColorLed 179  
get\_functionDescriptor, YCompass 216  
get\_functionDescriptor, YCurrent 260  
get\_functionId, YDataLogger 304  
get\_functionId, YDataSet 337  
get\_functionId, YDigitalIO 378  
get\_functionId, YDisplay 429  
get\_functionId, YDualPower 504  
get\_functionId, YFiles 537  
get\_functionId, YGenericSensor 569  
get\_functionId, YGyro 619  
get\_functionId, YHubPort 672  
get\_functionId, YHumidity 703  
get\_functionId, YLed 743  
get\_functionId, YLightSensor 778  
get\_functionId, YMagnetometer 821  
get\_functionId, YNetwork 926  
get\_functionId, YOsControl 976  
get\_functionId, YPower 1008  
get\_functionId, YPressure 1054  
get\_functionId, YPwmOutput 1100  
get\_functionId, YPwmPowerSource 1135  
get\_functionId, YQt 1166  
get\_functionId, YRealTimeClock 1206  
get\_functionId, YRefFrame 1244  
get\_functionId, YRelay 1279  
get\_functionId,YSensor 1320  
get\_functionId,YServo 1362  
get\_functionId,YTemperature 1402  
get\_functionId,YTilt 1447  
get\_functionId,YVoc 1490  
get\_functionId,YVoltage 1533  
get\_functionId,YVSource 1574  
get\_functionId,YWakeUpMonitor 1609  
get\_functionId,YWakeUpSchedule 1648  
get\_functionId,YWatchdog 1692  
get\_functionId,YWireless 1740  
get\_hardwareId, YAccelerometer 53  
get\_hardwareId, YAnButton 100  
get\_hardwareId, YCarbonDioxide 141  
get\_hardwareId, YColorLed 180  
get\_hardwareId, YCompass 217  
get\_hardwareId, YCurrent 261  
get\_hardwareId, YDataLogger 305  
get\_hardwareId, YDataSet 338  
get\_hardwareId, YDigitalIO 379  
get\_hardwareId, YDisplay 430  
get\_hardwareId, YDualPower 505  
get\_hardwareId, YFiles 538  
get\_hardwareId, YGenericSensor 570  
get\_hardwareId, YGyro 620  
get\_hardwareId, YHubPort 673  
get\_hardwareId, YHumidity 704  
get\_hardwareId, YLed 744  
get\_hardwareId, YLightSensor 779  
get\_hardwareId, YMagnetometer 822  
get\_hardwareId, YModule 874  
get\_hardwareId, YNetwork 927  
get\_hardwareId, YOsControl 977  
get\_hardwareId, YPower 1009  
get\_hardwareId, YPressure 1055  
get\_hardwareId, YPwmOutput 1101

get\_hardwareId, YPwmPowerSource 1136  
get\_hardwareId, YQt 1167  
get\_hardwareId, YRealTimeClock 1207  
get\_hardwareId, YRefFrame 1245  
get\_hardwareId, YRelay 1280  
get\_hardwareId,YSensor 1321  
get\_hardwareId,YServo 1363  
get\_hardwareId,YTemperature 1403  
get\_hardwareId, YTilt 1448  
get\_hardwareId, YVoc 1491  
get\_hardwareId, YVoltage 1534  
get\_hardwareId, YVSource 1575  
get\_hardwareId, YWakeUpMonitor 1610  
get\_hardwareId, YWakeUpSchedule 1649  
get\_hardwareId, YWatchdog 1693  
get\_hardwareId, YWireless 1741  
get\_heading, YGyro 621  
get\_highestValue, YAccelerometer 54  
get\_highestValue, YCarbonDioxide 142  
get\_highestValue, YCompass 218  
get\_highestValue, YCurrent 262  
get\_highestValue, YGenericSensor 571  
get\_highestValue, YGyro 622  
get\_highestValue, YHumidity 705  
get\_highestValue, YLightSensor 780  
get\_highestValue, YMagnetometer 823  
get\_highestValue, YPower 1010  
get\_highestValue, YPressure 1056  
get\_highestValue, YQt 1168  
get\_highestValue, YSensor 1322  
get\_highestValue, YTemperature 1404  
get\_highestValue, YTilt 1449  
get\_highestValue, YVoc 1492  
get\_highestValue, YVoltage 1535  
get\_hours, YWakeUpSchedule 1650  
get\_hslColor, YColorLed 181  
get\_icon2d, YModule 875  
get\_ipAddress, YNetwork 928  
get\_isPressed, YAnButton 101  
get\_lastLogs, YModule 876  
get\_lastTimePressed, YAnButton 102  
get\_lastTimeReleased, YAnButton 103  
get\_layerCount, YDisplay 431  
get\_layerHeight, YDisplay 432  
get\_layerHeight, YDisplayLayer 477  
get\_layerWidth, YDisplay 433  
get\_layerWidth, YDisplayLayer 478  
get\_linkQuality, YWireless 1742  
get\_list, YFiles 539  
get\_logFrequency, YAccelerometer 55  
get\_logFrequency, YCarbonDioxide 143  
get\_logFrequency, YCompass 219  
get\_logFrequency, YCurrent 263  
get\_logFrequency, YGenericSensor 572  
get\_logFrequency, YGyro 623  
get\_logFrequency, YHumidity 706  
get\_logFrequency, YLightSensor 781  
get\_logFrequency, YMagnetometer 824  
get\_logFrequency, YPower 1011  
get\_logFrequency, YPressure 1057  
get\_logFrequency, YQt 1169  
get\_logFrequency, YSensor 1323  
get\_logFrequency, YTemperature 1405  
get\_logFrequency, YTilt 1450  
get\_logFrequency, YVoc 1493  
get\_logFrequency, YVoltage 1536  
get\_logicalName, YAccelerometer 56  
get\_logicalName, YAnButton 104  
get\_logicalName, YCarbonDioxide 144  
get\_logicalName, YColorLed 182  
get\_logicalName, YCompass 220  
get\_logicalName, YCurrent 264  
get\_logicalName, YDataLogger 306  
get\_logicalName, YDigitalIO 380  
get\_logicalName, YDisplay 434  
get\_logicalName, YDualPower 506  
get\_logicalName, YFiles 540  
get\_logicalName, YGenericSensor 573  
get\_logicalName, YGyro 624  
get\_logicalName, YHubPort 674  
get\_logicalName, YHumidity 707  
get\_logicalName, YLed 745  
get\_logicalName, YLightSensor 782  
get\_logicalName, YMagnetometer 825  
get\_logicalName, YModule 877  
get\_logicalName, YNetwork 929  
get\_logicalName, YOsControl 978  
get\_logicalName, YPower 1012  
get\_logicalName, YPressure 1058  
get\_logicalName, YPwmOutput 1102  
get\_logicalName, YPwmPowerSource 1137  
get\_logicalName, YQt 1170  
get\_logicalName, YRealTimeClock 1208  
get\_logicalName, YRefFrame 1246  
get\_logicalName, YRelay 1281  
get\_logicalName, YSensor 1324  
get\_logicalName, YServo 1364  
get\_logicalName, YTemperature 1406  
get\_logicalName, YTilt 1451  
get\_logicalName, YVoc 1494  
get\_logicalName, YVoltage 1537  
get\_logicalName, YVSource 1576  
get\_logicalName, YWakeUpMonitor 1611  
get\_logicalName, YWakeUpSchedule 1651  
get\_logicalName, YWatchdog 1694  
get\_logicalName, YWireless 1743  
get\_lowestValue, YAccelerometer 57  
get\_lowestValue, YCarbonDioxide 145  
get\_lowestValue, YCompass 221  
get\_lowestValue, YCurrent 265  
get\_lowestValue, YGenericSensor 574  
get\_lowestValue, YGyro 625  
get\_lowestValue, YHumidity 708  
get\_lowestValue, YLightSensor 783  
get\_lowestValue, YMagnetometer 826  
get\_lowestValue, YPower 1013  
get\_lowestValue, YPressure 1059  
get\_lowestValue, YQt 1171

get\_lowestValue, YSensor 1325  
get\_lowestValue, YTemperature 1407  
get\_lowestValue, YTilt 1452  
get\_lowestValue, YVoc 1495  
get\_lowestValue, YVoltage 1538  
get\_luminosity, YLed 746  
get\_luminosity, YModule 878  
get\_macAddress, YNetwork 930  
get\_magneticHeading, YCompass 222  
get\_maxTimeOnStateA, YRelay 1282  
get\_maxTimeOnStateA, YWatchdog 1695  
get\_maxTimeOnStateB, YRelay 1283  
get\_maxTimeOnStateB, YWatchdog 1696  
get\_maxValue, YDataRun 327  
get\_maxValue, YDataStream 356  
get\_maxValue, YMeasure 856  
get\_measureNames, YDataRun 328  
get\_measures, YDataSet 339  
get\_message, YWireless 1744  
get\_meter, YPower 1014  
get\_meterTimer, YPower 1015  
get\_minutes, YWakeUpSchedule 1652  
get\_minutesA, YWakeUpSchedule 1653  
get\_minutesB, YWakeUpSchedule 1654  
get\_minValue, YDataRun 329  
get\_minValue, YDataStream 357  
get\_minValue, YMeasure 857  
get\_module, YAccelerometer 58  
get\_module, YAnButton 105  
get\_module, YCarbonDioxide 146  
get\_module, YColorLed 183  
get\_module, YCompass 223  
get\_module, YCurrent 266  
get\_module, YDataLogger 307  
get\_module, YDigitalIO 381  
get\_module, YDisplay 435  
get\_module, YDualPower 507  
get\_module, YFiles 541  
get\_module, YGenericSensor 575  
get\_module, YGyro 626  
get\_module, YHubPort 675  
get\_module, YHumidity 709  
get\_module, YLed 747  
get\_module, YLightSensor 784  
get\_module, YMagnetometer 827  
get\_module, YNetwork 931  
get\_module, YOsControl 979  
get\_module, YPower 1016  
get\_module, YPressure 1060  
get\_module, YPwmOutput 1103  
get\_module, YPwmPowerSource 1138  
get\_module, YQt 1172  
get\_module, YRealTimeClock 1209  
get\_module, YRefFrame 1247  
get\_module, YRelay 1284  
get\_module, YSensor 1326  
get\_module,YServo 1365  
get\_module, YTemperature 1408  
get\_module, YTilt 1453  
get\_module, YVoc 1496  
get\_module, YVoltage 1539  
get\_module, YVSource 1577  
get\_module, YWakeUpMonitor 1612  
get\_module, YWakeUpSchedule 1655  
get\_module, YWatchdog 1697  
get\_module, YWireless 1745  
get\_module\_async, YAccelerometer 59  
get\_module\_async, YAnButton 106  
get\_module\_async, YCarbonDioxide 147  
get\_module\_async, YColorLed 184  
get\_module\_async, YCompass 224  
get\_module\_async, YCurrent 267  
get\_module\_async, YDataLogger 308  
get\_module\_async, YDigitalIO 382  
get\_module\_async, YDisplay 436  
get\_module\_async, YDualPower 508  
get\_module\_async, YFiles 542  
get\_module\_async, YGenericSensor 576  
get\_module\_async, YGyro 627  
get\_module\_async, YHubPort 676  
get\_module\_async, YHumidity 710  
get\_module\_async, YLed 748  
get\_module\_async, YLightSensor 785  
get\_module\_async, YMagnetometer 828  
get\_module\_async, YNetwork 932  
get\_module\_async, YOsControl 980  
get\_module\_async, YPower 1017  
get\_module\_async, YPressure 1061  
get\_module\_async, YPwmOutput 1104  
get\_module\_async, YPwmPowerSource 1139  
get\_module\_async, YQt 1173  
get\_module\_async, YRealTimeClock 1210  
get\_module\_async, YRefFrame 1248  
get\_module\_async, YRelay 1285  
get\_module\_async, YSensor 1327  
get\_module\_async, YServo 1366  
get\_module\_async, YTemperature 1409  
get\_module\_async, YTilt 1454  
get\_module\_async, YVoc 1497  
get\_module\_async, YVoltage 1540  
get\_module\_async, YVSource 1578  
get\_module\_async, YWakeUpMonitor 1613  
get\_module\_async, YWakeUpSchedule 1656  
get\_module\_async, YWatchdog 1698  
get\_module\_async, YWireless 1746  
get\_monthDays, YWakeUpSchedule 1657  
get\_months, YWakeUpSchedule 1658  
get\_mountOrientation, YRefFrame 1249  
get\_mountPosition, YRefFrame 1250  
get\_neutral, YServo 1367  
get\_nextOccurrence, YWakeUpSchedule 1659  
get\_nextWakeUp, YWakeUpMonitor 1614  
get\_orientation, YDisplay 437  
get\_output, YRelay 1286  
get\_output, YWatchdog 1699  
get\_outputVoltage, YDigitalIO 383  
get\_overCurrent, YVSource 1579  
get\_overHeat, YVSource 1580

get\_overLoad, YVSource 1581  
get\_period, YPwmOutput 1105  
get\_persistentSettings, YModule 879  
get\_pitch, YGyro 628  
get\_poeCurrent, YNetwork 933  
get\_portDirection, YDigitalIO 384  
get\_portOpenDrain, YDigitalIO 385  
get\_portPolarity, YDigitalIO 386  
get\_portSize, YDigitalIO 387  
get\_portState, YDigitalIO 388  
get\_portState, YHubPort 677  
get\_position,YServo 1368  
get\_positionAtPowerOn,YServo 1369  
get\_power, YLed 749  
get\_powerControl, YDualPower 509  
get\_powerDuration, YWakeUpMonitor 1615  
get\_powerMode, YPwmPowerSource 1140  
get\_powerState, YDualPower 510  
get\_preview, YDataSet 340  
get\_primaryDNS, YNetwork 934  
get\_productId, YModule 880  
get\_productName, YModule 881  
get\_productRelease, YModule 882  
get\_progress, YDataSet 341  
get\_pulseCounter, YAnButton 107  
get\_pulseDuration, YPwmOutput 1106  
get\_pulseTimer, YAnButton 108  
get\_pulseTimer, YRelay 1287  
get\_pulseTimer, YWatchdog 1700  
get\_quaternionW, YGyro 629  
get\_quaternionX, YGyro 630  
get\_quaternionY, YGyro 631  
get\_quaternionZ, YGyro 632  
get\_range,YServo 1370  
get\_rawValue, YAnButton 109  
get\_readiness, YNetwork 935  
get\_rebootCountdown, YModule 883  
get\_recordedData, YAccelerometer 60  
get\_recordedData, YCarbonDioxide 148  
get\_recordedData, YCompass 225  
get\_recordedData, YCurrent 268  
get\_recordedData, YGenericSensor 577  
get\_recordedData, YGyro 633  
get\_recordedData, YHumidity 711  
get\_recordedData, YLightSensor 786  
get\_recordedData, YMagnetometer 829  
get\_recordedData, YPower 1018  
get\_recordedData, YPressure 1062  
get\_recordedData, YQt 1174  
get\_recordedData, YSensor 1328  
get\_recordedData, YTemperature 1410  
get\_recordedData, YTilt 1455  
get\_recordedData, YVoc 1498  
get\_recordedData, YVoltage 1541  
get\_recording, YDataLogger 309  
get\_regulationFailure, YVSource 1582  
get\_reportFrequency, YAccelerometer 61  
get\_reportFrequency, YCarbonDioxide 149  
get\_reportFrequency, YCompass 226  
get\_reportFrequency, YCurrent 269  
get\_reportFrequency, YGenericSensor 578  
get\_reportFrequency, YGyro 634  
get\_reportFrequency, YHumidity 712  
get\_reportFrequency, YLightSensor 787  
get\_reportFrequency, YMagnetometer 830  
get\_reportFrequency, YPower 1019  
get\_reportFrequency, YPressure 1063  
get\_reportFrequency, YQt 1175  
get\_reportFrequency, YSensor 1329  
get\_reportFrequency, YTemperature 1411  
get\_reportFrequency, YTilt 1456  
get\_reportFrequency, YVoc 1499  
get\_reportFrequency, YVoltage 1542  
get\_resolution, YAccelerometer 62  
get\_resolution, YCarbonDioxide 150  
get\_resolution, YCompass 227  
get\_resolution, YCurrent 270  
get\_resolution, YGenericSensor 579  
get\_resolution, YGyro 635  
get\_resolution, YHumidity 713  
get\_resolution, YLightSensor 788  
get\_resolution, YMagnetometer 831  
get\_resolution, YPower 1020  
get\_resolution, YPressure 1064  
get\_resolution, YQt 1176  
get\_resolution, YSensor 1330  
get\_resolution, YTemperature 1412  
get\_resolution, YTilt 1457  
get\_resolution, YVoc 1500  
get\_resolution, YVoltage 1543  
get\_rgbColor, YColorLed 185  
get\_rgbColorAtPowerOn, YColorLed 186  
get\_roll, YGyro 636  
get\_router, YNetwork 936  
getRowCount, YDataStream 358  
get\_runIndex, YDataStream 359  
get\_running, YWatchdog 1701  
get\_secondaryDNS, YNetwork 937  
get\_security, YWireless 1747  
get\_sensitivity, YAnButton 110  
get\_sensorType, YTemperature 1413  
get\_serialNumber, YModule 884  
get\_shutdownCountdown, YOsControl 981  
get\_signalRange, YGenericSensor 580  
get\_signalUnit, YGenericSensor 581  
get\_signalValue, YGenericSensor 582  
get\_sleepCountdown, YWakeUpMonitor 1616  
get\_ssId, YWireless 1748  
get\_startTime, YDataStream 360  
get\_startTimeUTC, YDataRun 330  
get\_startTimeUTC, YDataSet 342  
get\_startTimeUTC, YDataStream 361  
get\_startTimeUTC, YMeasure 858  
get\_startupSeq, YDisplay 438  
get\_state, YRelay 1288  
get\_state, YWatchdog 1702  
get\_stateAtPowerOn, YRelay 1289  
get\_stateAtPowerOn, YWatchdog 1703

get\_subnetMask, YNetwork 938  
get\_summary, YDataSet 343  
get\_timeSet, YRealTimeClock 1211  
get\_timeUTC, YDataLogger 310  
get\_triggerDelay, YWatchdog 1704  
get\_triggerDuration, YWatchdog 1705  
get\_unit, YAccelerometer 63  
get\_unit, YCarbonDioxide 151  
get\_unit, YCompass 228  
get\_unit, YCurrent 271  
get\_unit, YDataSet 344  
get\_unit, YGenericSensor 583  
get\_unit, YGyro 637  
get\_unit, YHumidity 714  
get\_unit, YLightSensor 789  
get\_unit, YMagnetometer 832  
get\_unit, YPower 1021  
get\_unit, YPressure 1065  
get\_unit, YQt 1177  
get\_unit,YSensor 1331  
get\_unit, YTTemperature 1414  
get\_unit, YTilt 1458  
get\_unit, YVoc 1501  
get\_unit, YVoltage 1544  
get\_unit, YVSource 1583  
get\_unixTime, YRealTimeClock 1212  
get\_upTime, YModule 885  
get\_usbBandwidth, YModule 886  
get\_usbCurrent, YModule 887  
get\_userData, YAccelerometer 64  
get\_userData, YAnButton 111  
get\_userData, YCarbonDioxide 152  
get\_userData, YColorLed 187  
get\_userData, YCompass 229  
get\_userData, YCurrent 272  
get\_userData, YDataLogger 311  
get\_userData, YDigitalIO 389  
get\_userData, YDisplay 439  
get\_userData, YDualPower 511  
get\_userData, YFiles 543  
get\_userData, YGenericSensor 584  
get\_userData, YGyro 638  
get\_userData, YHubPort 678  
get\_userData, YHumidity 715  
get\_userData, YLed 750  
get\_userData, YLightSensor 790  
get\_userData, YMagnetometer 833  
get\_userData, YModule 888  
get\_userData, YNetwork 939  
get\_userData, YOsControl 982  
get\_userData, YPower 1022  
get\_userData, YPressure 1066  
get\_userData, YPwmOutput 1107  
get\_userData, YPwmPowerSource 1141  
get\_userData, YQt 1178  
get\_userData, YRealTimeClock 1213  
get\_userData, YRefFrame 1251  
get\_userData, YRelay 1290  
get\_userData, YSensor 1332

get(userData, YServo 1371  
get(userData, YTTemperature 1415  
get(userData, YTilt 1459  
get(userData, YVoc 1502  
get(userData, YVoltage 1545  
get(userData, YVSource 1584  
get(userData, YWakeUpMonitor 1617  
get(userData, YWakeUpSchedule 1660  
get(userData, YWatchdog 1706  
get(userData, YWireless 1749  
get(userPassword, YNetwork 940  
get\_utcOffset, YRealTimeClock 1214  
get\_valueCount, YDataRun 331  
get\_valueInterval, YDataRun 332  
get\_valueRange, YGenericSensor 585  
get\_voltage, YVSource 1585  
get\_wakeUpReason, YWakeUpMonitor 1618  
get\_wakeUpState, YWakeUpMonitor 1619  
get\_weekDays, YWakeUpSchedule 1661  
get\_wwwWatchdogDelay, YNetwork 941  
get\_xValue, YAccelerometer 65  
get\_xValue, YGyro 639  
get\_xValue, YMagnetometer 834  
get\_yValue, YAccelerometer 66  
get\_yValue, YGyro 640  
get\_yValue, YMagnetometer 835  
get\_zValue, YAccelerometer 67  
get\_zValue, YGyro 641  
get\_zValue, YMagnetometer 836  
GetAPIVersion, YAPI 19  
GetTickCount, YAPI 20  
Gyro 606

## H

HandleEvents, YAPI 21  
hide, YDisplayLayer 479  
Horloge 1196  
hslMove, YColorLed 188  
Humidity 690

## I

InitAPI, YAPI 22  
Installation 3  
Interface 39, 85, 127, 170, 203, 247, 290, 363, 411, 462, 494, 523, 556, 606, 661, 690, 733, 764, 808, 860, 906, 994, 1041, 1084, 1126, 1153, 1196, 1267, 1307, 1350, 1389, 1434, 1477, 1520, 1563, 1599, 1638, 1679, 1728  
Introduction 1  
isOnline, YAccelerometer 68  
isOnline, YAnButton 112  
isOnline, YCarbonDioxide 153  
isOnline, YColorLed 189  
isOnline, YCompass 230  
isOnline, YCurrent 273  
isOnline, YDataLogger 312  
isOnline, YDigitalIO 390  
isOnline, YDisplay 440

isOnline, YDualPower 512  
isOnline, YFiles 544  
isOnline, YGenericSensor 586  
isOnline, YGyro 642  
isOnline, YHubPort 679  
isOnline, YHumidity 716  
isOnline, YLed 751  
isOnline, YLightSensor 791  
isOnline, YMagnetometer 837  
isOnline, YModule 889  
isOnline, YNetwork 942  
isOnline, YOsControl 983  
isOnline, YPower 1023  
isOnline, YPressure 1067  
isOnline, YPwmOutput 1108  
isOnline, YPwmPowerSource 1142  
isOnline, YQt 1179  
isOnline, YRealTimeClock 1215  
isOnline, YRefFrame 1252  
isOnline, YRelay 1292  
isOnline, YSensor 1334  
isOnline, YServo 1373  
isOnline\_async, YTTemperature 1417  
isOnline\_async, YTilt 1461  
isOnline\_async, YVoc 1504  
isOnline\_async, YVoltage 1547  
isOnline\_async, YVSource 1587  
isOnline\_async, YWakeUpMonitor 1621  
isOnline\_async, YWakeUpSchedule 1663  
isOnline\_async, YWatchdog 1708  
isOnline\_async, YWireless 1751

**J**

joinNetwork, YWireless 1752

**L**

LightSensor 764  
lineTo, YDisplayLayer 480  
load, YAccelerometer 70  
load, YAnButton 114  
load, YCarbonDioxide 155  
load, YColorLed 191  
load, YCompass 232  
load, YCurrent 275  
load, YDataLogger 314  
load, YDigitalIO 392  
load, YDisplay 442  
load, YDualPower 514  
load, YFiles 546  
load, YGenericSensor 588  
load, YGyro 644  
load, YHubPort 681  
load, YHumidity 718  
load, YLed 753  
load, YLightSensor 793  
load, YMagnetometer 839  
load, YModule 891  
load, YNetwork 944  
load, YOsControl 985  
load, YPower 1025  
load, YPressure 1069  
load, YPwmOutput 1110  
load, YPwmPowerSource 1144  
load, YQt 1181  
load, YRealTimeClock 1217  
load, YRefFrame 1254  
load, YRelay 1293  
load, YSensor 1335  
load, YServo 1374  
load, YTTemperature 1418  
load, YTilt 1462  
load, YVoc 1505  
load, YVoltage 1548  
load, YVSource 1588  
load, YWakeUpMonitor 1622  
load, YWakeUpSchedule 1664  
load, YWatchdog 1709

load, YWireless 1753  
load\_async, YAccelerometer 72  
load\_async, YAnButton 115  
load\_async, YCarbonDioxide 157  
load\_async, YColorLed 192  
load\_async, YCompass 234  
load\_async, YCurrent 277  
load\_async, YDataLogger 315  
load\_async, YDigitalIO 393  
load\_async, YDisplay 443  
load\_async, YDualPower 515  
load\_async, YFiles 547  
load\_async, YGenericSensor 590  
load\_async, YGyro 646  
load\_async, YHubPort 682  
load\_async, YHumidity 720  
load\_async, YLed 754  
load\_async, YLightSensor 795  
load\_async, YMagnetometer 841  
load\_async, YModule 892  
load\_async, YNetwork 945  
load\_async, YOsControl 986  
load\_async, YPower 1027  
load\_async, YPressure 1071  
load\_async, YPwmOutput 1111  
load\_async, YPwmPowerSource 1145  
load\_async, YQt 1183  
load\_async, YRealTimeClock 1218  
load\_async, YRefFrame 1255  
load\_async, YRelay 1294  
load\_async, YSensor 1337  
load\_async, YServo 1375  
load\_async, YTemperature 1420  
load\_async, YTilt 1464  
load\_async, YVoc 1507  
load\_async, YVoltage 1550  
load\_async, YVSource 1589  
load\_async, YWakeUpMonitor 1623  
load\_async, YWakeUpSchedule 1665  
load\_async, YWatchdog 1710  
load\_async, YWireless 1754  
loadCalibrationPoints, YAccelerometer 71  
loadCalibrationPoints, YCarbonDioxide 156  
loadCalibrationPoints, YCompass 233  
loadCalibrationPoints, YCurrent 276  
loadCalibrationPoints, YGenericSensor 589  
loadCalibrationPoints, YGyro 645  
loadCalibrationPoints, YHumidity 719  
loadCalibrationPoints, YLightSensor 794  
loadCalibrationPoints, YMagnetometer 840  
loadCalibrationPoints, YPower 1026  
loadCalibrationPoints, YPressure 1070  
loadCalibrationPoints, YQt 1182  
loadCalibrationPoints, YSensor 1336  
loadCalibrationPoints, YTemperature 1419  
loadCalibrationPoints, YTilt 1463  
loadCalibrationPoints, YVoc 1506  
loadCalibrationPoints, YVoltage 1549  
loadMore, YDataSet 345

loadMore\_async, YDataSet 346

## M

Magnetometer 808  
Mesurée 854  
Mise 325  
Module 6, 860  
more3DCalibration, YRefFrame 1256  
move,YServo 1376  
moveTo, YDisplayLayer 481

## N

Network 906  
newSequence, YDisplay 444  
nextAccelerometer, YAccelerometer 73  
nextAnButton, YAnButton 116  
nextCarbonDioxide, YCarbonDioxide 158  
nextColorLed, YColorLed 193  
nextCompass, YCompass 235  
nextCurrent, YCurrent 278  
nextDataLogger, YDataLogger 316  
nextDigitalIO, YDigitalIO 394  
nextDisplay, YDisplay 445  
nextDualPower, YDualPower 516  
nextFiles, YFiles 548  
nextGenericSensor, YGenericSensor 591  
nextGyro, YGyro 647  
nextHubPort, YHubPort 683  
nextHumidity, YHumidity 721  
nextLed, YLed 755  
nextLightSensor, YLightSensor 796  
nextMagnetometer, YMagnetometer 842  
nextModule, YModule 893  
nextNetwork, YNetwork 946  
nextOsControl, YOsControl 987  
nextPower, YPower 1028  
nextPressure, YPressure 1072  
nextPwmOutput, YPwmOutput 1112  
nextPwmPowerSource, YPwmPowerSource 1146  
nextQt, YQt 1184  
nextRealTimeClock, YRealTimeClock 1219  
nextRefFrame, YRefFrame 1257  
nextRelay, YRelay 1295  
nextSensor, YSensor 1338  
nextServo, YServo 1377  
nextTemperature, YTemperature 1421  
nextTilt, YTilt 1465  
nextVoc, YVoc 1508  
nextVoltage, YVoltage 1551  
nextVSource, YVSource 1590  
nextWakeUpMonitor, YWakeUpMonitor 1624  
nextWakeUpSchedule, YWakeUpSchedule 1666  
nextWatchdog, YWatchdog 1711  
nextWireless, YWireless 1755

## O

Objets 462

## P

pauseSequence, YDisplay 446  
ping, YNetwork 947  
playSequence, YDisplay 447  
Port 661  
Power 994  
PreregisterHub, YAPI 23  
Pressure 1041  
Projet 3  
pulse, YDigitalIO 395  
pulse, YRelay 1296  
pulse, YVSource 1591  
pulse, YWatchdog 1712  
pulseDurationMove, YPwmOutput 1113  
PwmPowerSource 1126

## Q

Quaternion 1153

## R

Real 1196  
reboot, YModule 894  
Reference 12  
Référentiel 1227  
registerAnglesCallback, YGyro 648  
RegisterDeviceArrivalCallback, YAPI 24  
RegisterDeviceRemovalCallback, YAPI 25  
RegisterHub, YAPI 26  
RegisterHubDiscoveryCallback, YAPI 28  
registerLogCallback, YModule 895  
RegisterLogFunction, YAPI 29  
registerQuaternionCallback, YGyro 649  
registerTimedReportCallback, YAccelerometer 74  
registerTimedReportCallback, YCarbonDioxide 159  
registerTimedReportCallback, YCompass 236  
registerTimedReportCallback, YCurrent 279  
registerTimedReportCallback, YGenericSensor 592  
registerTimedReportCallback, YGyro 650  
registerTimedReportCallback, YHumidity 722  
registerTimedReportCallback, YLightSensor 797  
registerTimedReportCallback, YMagnetometer 843  
registerTimedReportCallback, YPower 1029  
registerTimedReportCallback, YPressure 1073  
registerTimedReportCallback, YQt 1185  
registerTimedReportCallback, YSensor 1339  
registerTimedReportCallback, YTemperature 1422  
registerTimedReportCallback, YTilt 1466  
registerTimedReportCallback, YVoc 1509  
registerTimedReportCallback, YVoltage 1552

registerValueCallback, YAccelerometer 75  
registerValueCallback, YAnButton 117  
registerValueCallback, YCarbonDioxide 160  
registerValueCallback, YColorLed 194  
registerValueCallback, YCompass 237  
registerValueCallback, YCurrent 280  
registerValueCallback, YDataLogger 317  
registerValueCallback, YDigitalIO 396  
registerValueCallback, YDisplay 448  
registerValueCallback, YDualPower 517  
registerValueCallback, YFiles 549  
registerValueCallback, YGenericSensor 593  
registerValueCallback, YGyro 651  
registerValueCallback, YHubPort 684  
registerValueCallback, YHumidity 723  
registerValueCallback, YLed 756  
registerValueCallback, YLightSensor 798  
registerValueCallback, YMagnetometer 844  
registerValueCallback, YNetwork 948  
registerValueCallback, YOsControl 988  
registerValueCallback, YPower 1030  
registerValueCallback, YPressure 1074  
registerValueCallback, YPwmOutput 1114  
registerValueCallback, YPwmPowerSource 1147  
registerValueCallback, YQt 1186  
registerValueCallback, YRealTimeClock 1220  
registerValueCallback, YRefFrame 1258  
registerValueCallback, YRelay 1297  
registerValueCallback, YSensor 1340  
registerValueCallback, YServo 1378  
registerValueCallback, YTemperature 1423  
registerValueCallback, YTilt 1467  
registerValueCallback, YVoc 1510  
registerValueCallback, YVoltage 1553  
registerValueCallback, YVSource 1592  
registerValueCallback, YWakeUpMonitor 1625  
registerValueCallback, YWakeUpSchedule 1667  
registerValueCallback, YWatchdog 1713  
registerValueCallback, YWireless 1756  
Relay 1267  
remove, YFiles 550  
reset, YDisplayLayer 482  
reset, YPower 1031  
resetAll, YDisplay 449  
resetCounter, YAnButton 118  
resetSleepCountDown, YWakeUpMonitor 1626  
resetWatchdog, YWatchdog 1714  
revertFromFlash, YModule 896  
rgbMove, YColorLed 195

## S

save3DCalibration, YRefFrame 1259  
saveSequence, YDisplay 450  
saveToFlash, YModule 897  
SelectArchitecture, YAPI 30  
selectColorPen, YDisplayLayer 483  
selectEraser, YDisplayLayer 484  
selectFont, YDisplayLayer 485  
selectGrayPen, YDisplayLayer 486

Senseur 1307  
Séquence 325, 335, 348  
Servo 1350  
set\_adminPassword, YNetwork 949  
set\_analogCalibration, YAnButton 119  
set\_autoStart, YDataLogger 318  
set\_autoStart, YWatchdog 1715  
set\_beacon, YModule 898  
set\_bearing, YRefFrame 1260  
set\_bitDirection, YDigitalIO 397  
set\_bitOpenDrain, YDigitalIO 398  
set\_bitPolarity, YDigitalIO 399  
set\_bitState, YDigitalIO 400  
set\_blinking, YLed 757  
set\_brightness, YDisplay 451  
set\_calibrationMax, YAnButton 120  
set\_calibrationMin, YAnButton 121  
set\_callbackCredentials, YNetwork 950  
set\_callbackEncoding, YNetwork 951  
set\_callbackMaxDelay, YNetwork 952  
set\_callbackMethod, YNetwork 953  
set\_callbackMinDelay, YNetwork 954  
set\_callbackUrl, YNetwork 955  
set\_discoverable, YNetwork 956  
set\_dutyCycle, YPwmOutput 1115  
set\_dutyCycleAtPowerOn, YPwmOutput 1116  
set\_enabled, YDisplay 452  
set\_enabled, YHubPort 685  
set\_enabled, YPwmOutput 1117  
set\_enabled, YServo 1379  
set\_enabledAtPowerOn, YPwmOutput 1118  
set\_enabledAtPowerOn, YServo 1380  
set\_frequency, YPwmOutput 1119  
set\_highestValue, YAccelerometer 76  
set\_highestValue, YCarbonDioxide 161  
set\_highestValue, YCompass 238  
set\_highestValue, YCurrent 281  
set\_highestValue, YGenericSensor 594  
set\_highestValue, YGyro 652  
set\_highestValue, YHumidity 724  
set\_highestValue, YLightSensor 799  
set\_highestValue, YMagnetometer 845  
set\_highestValue, YPower 1032  
set\_highestValue, YPressure 1075  
set\_highestValue, YQt 1187  
set\_highestValue, YSensor 1341  
set\_highestValue, YTTemperature 1424  
set\_highestValue, YTilt 1468  
set\_highestValue, YVoc 1511  
set\_highestValue, YVoltage 1554  
set\_hours, YWakeUpSchedule 1668  
set\_hslColor, YColorLed 196  
set\_logFrequency, YAccelerometer 77  
set\_logFrequency, YCarbonDioxide 162  
set\_logFrequency, YCompass 239  
set\_logFrequency, YCurrent 282  
set\_logFrequency, YGenericSensor 595  
set\_logFrequency, YGyro 653  
set\_logFrequency, YHumidity 725  
set\_logFrequency, YLightSensor 800  
set\_logFrequency, YMagnetometer 846  
set\_logFrequency, YPower 1033  
set\_logFrequency, YPressure 1076  
set\_logFrequency, YQt 1188  
set\_logFrequency, YSensor 1342  
set\_logFrequency, YTTemperature 1425  
set\_logFrequency, YTilt 1469  
set\_logFrequency, YVoc 1512  
set\_logFrequency, YVoltage 1555  
set\_logicalName, YAccelerometer 78  
set\_logicalName, YAnButton 122  
set\_logicalName, YCarbonDioxide 163  
set\_logicalName, YColorLed 197  
set\_logicalName, YCompass 240  
set\_logicalName, YCurrent 283  
set\_logicalName, YDataLogger 319  
set\_logicalName, YDigitalIO 401  
set\_logicalName, YDisplay 453  
set\_logicalName, YDualPower 518  
set\_logicalName, YFiles 551  
set\_logicalName, YGenericSensor 596  
set\_logicalName, YGyro 654  
set\_logicalName, YHubPort 686  
set\_logicalName, YHumidity 726  
set\_logicalName, YLed 758  
set\_logicalName, YLightSensor 801  
set\_logicalName, YMagnetometer 847  
set\_logicalName, YModule 899  
set\_logicalName, YNetwork 957  
set\_logicalName, YOsControl 989  
set\_logicalName, YPower 1034  
set\_logicalName, YPressure 1077  
set\_logicalName, YPwmOutput 1120  
set\_logicalName, YPwmPowerSource 1148  
set\_logicalName, YQt 1189  
set\_logicalName, YRealTimeClock 1221  
set\_logicalName, YRefFrame 1261  
set\_logicalName, YRelay 1298  
set\_logicalName, YSensor 1343  
set\_logicalName, YServo 1381  
set\_logicalName, YTTemperature 1426  
set\_logicalName, YTilt 1470  
set\_logicalName, YVoc 1513  
set\_logicalName, YVoltage 1556  
set\_logicalName, YVSource 1593  
set\_logicalName, YWakeUpMonitor 1627  
set\_logicalName, YWakeUpSchedule 1669  
set\_logicalName, YWatchdog 1716  
set\_logicalName, YWireless 1757  
set\_lowestValue, YAccelerometer 79  
set\_lowestValue, YCarbonDioxide 164  
set\_lowestValue, YCompass 241  
set\_lowestValue, YCurrent 284  
set\_lowestValue, YGenericSensor 597  
set\_lowestValue, YGyro 655  
set\_lowestValue, YHumidity 727  
set\_lowestValue, YLightSensor 802  
set\_lowestValue, YMagnetometer 848

set\_lowestValue, YPower 1035  
set\_lowestValue, YPressure 1078  
set\_lowestValue, YQt 1190  
set\_lowestValue, YSensor 1344  
set\_lowestValue, YTemperature 1427  
set\_lowestValue, YTilt 1471  
set\_lowestValue, YVoc 1514  
set\_lowestValue, YVoltage 1557  
set\_luminosity, YLed 759  
set\_luminosity, YModule 900  
set\_maxTimeOnStateA, YRelay 1299  
set\_maxTimeOnStateA, YWatchdog 1717  
set\_maxTimeOnStateB, YRelay 1300  
set\_maxTimeOnStateB, YWatchdog 1718  
set\_minutes, YWakeUpSchedule 1670  
set\_minutesA, YWakeUpSchedule 1671  
set\_minutesB, YWakeUpSchedule 1672  
set\_monthDays, YWakeUpSchedule 1673  
set\_months, YWakeUpSchedule 1674  
set\_mountPosition, YRefFrame 1262  
set\_neutral,YServo 1382  
set\_nextWakeUp, YWakeUpMonitor 1628  
set\_orientation, YDisplay 454  
set\_output, YRelay 1301  
set\_output, YWatchdog 1719  
set\_outputVoltage, YDigitalIO 402  
set\_period, YPwmOutput 1121  
set\_portDirection, YDigitalIO 403  
set\_portOpenDrain, YDigitalIO 404  
set\_portPolarity, YDigitalIO 405  
set\_portState, YDigitalIO 406  
set\_position, YServo 1383  
set\_positionAtPowerOn, YServo 1384  
set\_power, YLed 760  
set\_powerControl, YDualPower 519  
set\_powerDuration, YWakeUpMonitor 1629  
set\_powerMode, YPwmPowerSource 1149  
set\_primaryDNS, YNetwork 958  
set\_pulseDuration, YPwmOutput 1122  
set\_range, YServo 1385  
set\_recording, YDataLogger 320  
set\_reportFrequency, YAccelerometer 80  
set\_reportFrequency, YCarbonDioxide 165  
set\_reportFrequency, YCompass 242  
set\_reportFrequency, YCurrent 285  
set\_reportFrequency, YGenericSensor 598  
set\_reportFrequency, YGyro 656  
set\_reportFrequency, YHumidity 728  
set\_reportFrequency, YLightSensor 803  
set\_reportFrequency, YMagnetometer 849  
set\_reportFrequency, YPower 1036  
set\_reportFrequency, YPressure 1079  
set\_reportFrequency, YQt 1191  
set\_reportFrequency, YSensor 1345  
set\_reportFrequency, YTemperature 1428  
set\_reportFrequency, YTilt 1472  
set\_reportFrequency, YVoc 1515  
set\_reportFrequency, YVoltage 1558  
set\_resolution, YAccelerometer 81  
set\_resolution, YCarbonDioxide 166  
set\_resolution, YCompass 243  
set\_resolution, YCurrent 286  
set\_resolution, YGenericSensor 599  
set\_resolution, YGyro 657  
set\_resolution, YHumidity 729  
set\_resolution, YLightSensor 804  
set\_resolution, YMagnetometer 850  
set\_resolution, YPower 1037  
set\_resolution, YPressure 1080  
set\_resolution, YQt 1192  
set\_resolution, YSensor 1346  
set\_resolution, YTemperature 1429  
set\_resolution, YTilt 1473  
set\_resolution, YVoc 1516  
set\_resolution, YVoltage 1559  
set\_rgbColor, YColorLed 198  
set\_rgbColorAtPowerOn, YColorLed 199  
set\_running, YWatchdog 1720  
set\_secondaryDNS, YNetwork 959  
set\_sensitivity, YAnButton 123  
set\_sensorType, YTemperature 1430  
set\_signalRange, YGenericSensor 600  
set\_sleepCountdown, YWakeUpMonitor 1630  
set\_startupSeq, YDisplay 455  
set\_state, YRelay 1302  
set\_state, YWatchdog 1721  
set\_stateAtPowerOn, YRelay 1303  
set\_stateAtPowerOn, YWatchdog 1722  
set\_timeUTC, YDataLogger 321  
set\_triggerDelay, YWatchdog 1723  
set\_triggerDuration, YWatchdog 1724  
set\_unit, YGenericSensor 601  
set\_unixTime, YRealTimeClock 1222  
set\_usbBandwidth, YModule 901  
set\_userData, YAccelerometer 82  
set\_userData, YAnButton 124  
set\_userData, YCarbonDioxide 167  
set\_userData, YColorLed 200  
set\_userData, YCompass 244  
set\_userData, YCurrent 287  
set\_userData, YDataLogger 322  
set\_userData, YDigitalIO 407  
set\_userData, YDisplay 456  
set\_userData, YDualPower 520  
set\_userData, YFiles 552  
set\_userData, YGenericSensor 602  
set\_userData, YGyro 658  
set\_userData, YHubPort 687  
set\_userData, YHumidity 730  
set\_userData, YLed 761  
set\_userData, YLightSensor 805  
set\_userData, YMagnetometer 851  
set\_userData, YModule 902  
set\_userData, YNetwork 960  
set\_userData, YOsControl 990  
set\_userData, YPower 1038  
set\_userData, YPressure 1081  
set\_userData, YPwmOutput 1123

set(userData, YPwmPowerSource 1150  
set(userData, YQt 1193  
set(userData, YRealTimeClock 1223  
set(userData, YRefFrame 1263  
set(userData, YRelay 1304  
set(userData, YSensor 1347  
set(userData, YServo 1386  
set(userData, YTemperature 1431  
set(userData, YTilt 1474  
set(userData, YVoc 1517  
set(userData, YVoltage 1560  
set(userData, YVSource 1594  
set(userData, YWakeUpMonitor 1631  
set(userData, YWakeUpSchedule 1675  
set(userData, YWatchdog 1725  
set(userData, YWireless 1758  
set(userPassword, YNetwork 961  
set(utcOffset, YRealTimeClock 1224  
set(valueInterval, YDataRun 333  
set(valueRange, YGenericSensor 603  
set(voltage, YVSource 1595  
set(weekDays, YWakeUpSchedule 1676  
set(wwwWatchdogDelay, YNetwork 962  
setAntialiasingMode, YDisplayLayer 487  
setConsoleBackground, YDisplayLayer 488  
setConsoleMargins, YDisplayLayer 489  
setConsoleWordWrap, YDisplayLayer 490  
SetDelegate, YAPI 31  
setLayerPosition, YDisplayLayer 491  
SetTimeout, YAPI 32  
shutdown, YOsControl 991  
Sleep, YAPI 33  
sleep, YWakeUpMonitor 1632  
sleepFor, YWakeUpMonitor 1633  
sleepUntil, YWakeUpMonitor 1634  
Source 1563  
start3DCalibration, YRefFrame 1264  
stopSequence, YDisplay 457  
swapLayerContent, YDisplay 458

## T

Temperature 1389  
Temps 1196  
Tension 1563  
Tilt 1434  
toggle\_bitState, YDigitalIO 408  
triggerFirmwareUpdate, YModule 903  
TriggerHubDiscovery, YAPI 34  
Type 1307

## U

unhide, YDisplayLayer 492  
UnregisterHub, YAPI 35  
UpdateDeviceList, YAPI 36  
UpdateDeviceList\_async, YAPI 37  
upload, YDisplay 459  
upload, YFiles 553  
useDHCP, YNetwork 963

useStaticIP, YNetwork 964

## V

Valeur 854  
Visual 3  
Voltage 1520  
voltageMove, YVSource 1596

## W

wait\_async, YAccelerometer 83  
wait\_async, YAnButton 125  
wait\_async, YCarbonDioxide 168  
wait\_async, YColorLed 201  
wait\_async, YCompass 245  
wait\_async, YCurrent 288  
wait\_async, YDataLogger 323  
wait\_async, YDigitalIO 409  
wait\_async, YDisplay 460  
wait\_async, YDualPower 521  
wait\_async, YFiles 554  
wait\_async, YGenericSensor 604  
wait\_async, YGyro 659  
wait\_async, YHubPort 688  
wait\_async, YHumidity 731  
wait\_async, YLed 762  
wait\_async, YLightSensor 806  
wait\_async, YMagnetometer 852  
wait\_async, YModule 904  
wait\_async, YNetwork 965  
wait\_async, YOsControl 992  
wait\_async, YPower 1039  
wait\_async, YPressure 1082  
wait\_async, YPwmOutput 1124  
wait\_async, YPwmPowerSource 1151  
wait\_async, YQt 1194  
wait\_async, YRealTimeClock 1225  
wait\_async, YRefFrame 1265  
wait\_async, YRelay 1305  
wait\_async, YSensor 1348  
wait\_async, YServo 1387  
wait\_async, YTemperature 1432  
wait\_async, YTilt 1475  
wait\_async, YVoc 1518  
wait\_async, YVoltage 1561  
wait\_async, YVSource 1597  
wait\_async, YWakeUpMonitor 1635  
wait\_async, YWakeUpSchedule 1677  
wait\_async, YWatchdog 1726  
wait\_async, YWireless 1759  
wakeUp, YWakeUpMonitor 1636  
WakeUpMonitor 1599  
WakeUpSchedule 1638  
Watchdog 1679  
Wireless 1728

## Y

YAccelerometer 41-83

YAnButton 87-125  
YAPI 28-37  
YCarbonDioxide 129-168  
yCheckLogicalName 14  
YColorLed 171-201  
YCompass 205-245  
YCurrent 249-288  
YDataLogger 291-323  
YDataRun 325-333  
YDataSet 336-346  
YDataStream 349-361  
YDigitalIO 365-409  
yDisableExceptions 15  
YDisplay 413-460  
YDisplayLayer 463-492  
YDualPower 495-521  
yEnableExceptions 16  
yEnableUSBHost 17  
YFiles 524-554  
yFindAccelerometer 41  
yFindAnButton 87  
yFindCarbonDioxide 129  
yFindColorLed 171  
yFindCompass 205  
yFindCurrent 249  
yFindDataLogger 291  
yFindDigitalIO 365  
yFindDisplay 413  
yFindDualPower 495  
yFindFiles 524  
yFindGenericSensor 558  
yFindGyro 608  
yFindHubPort 662  
yFindHumidity 692  
yFindLed 734  
yFindLightSensor 766  
yFindMagnetometer 810  
yFindModule 862  
yFindNetwork 909  
yFindOsControl 968  
yFindPower 996  
yFindPressure 1043  
yFindPwmOutput 1086  
yFindPwmPowerSource 1127  
yFindQt 1155  
yFindRealTimeClock 1197  
yFindRefFrame 1229  
yFindRelay 1269  
yFindSensor 1309  
yFindServo 1352  
yFindTemperature 1391  
yFindTilt 1436  
yFindVoc 1479  
yFindVoltage 1522  
yFindVSource 1564  
yFindWakeUpMonitor 1601  
yFindWakeUpSchedule 1640  
yFindWatchdog 1681  
yFindWireless 1729  
yFirstAccelerometer 42  
yFirstAnButton 88  
yFirstCarbonDioxide 130  
yFirstColorLed 172  
yFirstCompass 206  
yFirstCurrent 250  
yFirstDataLogger 292  
yFirstDigitalIO 366  
yFirstDisplay 414  
yFirstDualPower 496  
yFirstFiles 525  
yFirstGenericSensor 559  
yFirstGyro 609  
yFirstHubPort 663  
yFirstHumidity 693  
yFirstLed 735  
yFirstLightSensor 767  
yFirstMagnetometer 811  
yFirstModule 863  
yFirstNetwork 910  
yFirstOsControl 969  
yFirstPower 997  
yFirstPressure 1044  
yFirstPwmOutput 1087  
yFirstPwmPowerSource 1128  
yFirstQt 1156  
yFirstRealTimeClock 1198  
yFirstRefFrame 1230  
yFirstRelay 1270  
yFirstSensor 1310  
yFirstServo 1353  
yFirstTemperature 1392  
yFirstTilt 1437  
yFirstVoc 1480  
yFirstVoltage 1523  
yFirstVSource 1565  
yFirstWakeUpMonitor 1602  
yFirstWakeUpSchedule 1641  
yFirstWatchdog 1682  
yFirstWireless 1730  
yFreeAPI 18  
YGenericSensor 558-604  
yGetAPIVersion 19  
yGetTickCount 20  
YGyro 608-659  
yHandleEvents 21  
YHubPort 662-688  
YHumidity 692-731  
yInitAPI 22  
YLed 734-762  
YLightSensor 766-806  
YMagnetometer 810-852  
YMeasure 854-858  
YModule 862-904  
YNetwork 909-965  
Yocto-Demo 3  
Yocto-hub 661  
YOsControl 968-992  
YPower 996-1039

yPreregisterHub 23  
YPressure 1043-1082  
YPwmOutput 1086-1124  
YPwmPowerSource 1127-1151  
YQt 1155-1194  
YRealTimeClock 1197-1225  
YRefFrame 1229-1265  
yRegisterDeviceArrivalCallback 24  
yRegisterDeviceRemovalCallback 25  
yRegisterHub 26  
yRegisterHubDiscoveryCallback 28  
yRegisterLogFunction 29  
YRelay 1269-1305  
ySelectArchitecture 30  
YSensor 1309-1348  
YServo 1352-1387

ySetDelegate 31  
ySetTimeout 32  
ySleep 33  
YTemperature 1391-1432  
YTilt 1436-1475  
yTriggerHubDiscovery 34  
yUnregisterHub 35  
yUpdateDeviceList 36  
yUpdateDeviceList\_async 37  
YVoc 1479-1518  
YVoltage 1522-1561  
YVSource 1564-1597  
YWakeUpMonitor 1601-1636  
YWakeUpSchedule 1640-1677  
YWatchdog 1681-1726  
YWireless 1729-1759