



## Référence de l'API VisualBasic .NET



# Table des matières

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Utilisation du Yocto-Demo en VisualBasic .NET .....</b>	<b>3</b>
2.1. Installation .....	3
2.2. Utilisation l'API yoctopuce dans un projet Visual Basic .....	3
2.3. Contrôle de la fonction Led .....	4
2.4. Contrôle de la partie module .....	6
2.5. Gestion des erreurs .....	8
Blueprint .....	10
<b>3. Reference .....</b>	<b>10</b>
3.1. Fonctions générales .....	11
3.2. Interface de la fonction Accelerometer .....	31
3.3. Interface de la fonction Altitude .....	72
3.4. Interface de la fonction AnButton .....	113
3.5. Interface de la fonction CarbonDioxide .....	150
3.6. Interface de la fonction ColorLed .....	188
3.7. Interface de la fonction Compass .....	216
3.8. Interface de la fonction Current .....	255
3.9. Interface de la fonction DataLogger .....	293
3.10. Séquence de données mise en forme .....	326
3.11. Séquence de données enregistrées .....	328
3.12. Séquence de données enregistrées brute .....	340
3.13. Interface de la fonction DigitalIO .....	355
3.14. Interface de la fonction Display .....	398
3.15. Interface des objets DisplayLayer .....	444
3.16. Interface de contrôle de l'alimentation .....	476
3.17. Interface de la fonction Files .....	500
3.18. Interface de la fonction GenericSensor .....	527
3.19. Interface de la fonction Gyro .....	575
3.20. Interface d'un port de Yocto-hub .....	625
3.21. Interface de la fonction Humidity .....	649
3.22. Interface de la fonction Led .....	687
3.23. Interface de la fonction LightSensor .....	713
3.24. Interface de la fonction Magnetometer .....	754

3.25. Valeur mesurée .....	795
3.26. Interface de contrôle du module .....	801
3.27. Interface de la fonction Motor .....	848
3.28. Interface de la fonction Network .....	888
3.29. contrôle d'OS .....	944
3.30. Interface de la fonction Power .....	966
3.31. Interface de la fonction Pressure .....	1008
3.32. Interface de la fonction PwmInput .....	1046
3.33. Interface de la fonction Pwm .....	1093
3.34. Interface de la fonction PwmPowerSource .....	1130
3.35. Interface du quaternion .....	1152
3.36. Interface de la fonction Horloge Temps Real .....	1190
3.37. Configuration du référentiel .....	1216
3.38. Interface de la fonction Relay .....	1251
3.39. Interface des fonctions de type senseur .....	1286
3.40. Interface de la fonction SerialPort .....	1324
3.41. Interface de la fonction Servo .....	1380
3.42. Interface de la fonction Temperature .....	1414
3.43. Interface de la fonction Tilt .....	1454
3.44. Interface de la fonction Voc .....	1492
3.45. Interface de la fonction Voltage .....	1530
3.46. Interface de la fonction Source de tension .....	1568
3.47. Interface de la fonction WakeUpMonitor .....	1599
3.48. Interface de la fonction WakeUpSchedule .....	1633
3.49. Interface de la fonction Watchdog .....	1669
3.50. Interface de la fonction Wireless .....	1713
<b>Index .....</b>	<b>1743</b>

# 1. Introduction

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

VisualBasic a longtemps été la porte d'entrée privilégiée vers le monde Microsoft. Nous nous devions donc d'offrir notre interface pour ce langage, même si la nouvelle tendance est le C#. Tous les exemples et les modèles de projet sont testés avec Microsoft Visual Basic 2010 Express, disponible gratuitement sur le site de Microsoft<sup>1</sup>.

### 2.1. Installation

Téléchargez la librairie Yoctopuce pour Visual Basic 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 Basic 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 Basic

La librairie Yoctopuce pour Visual Basic .NET se présente sous la forme d'une DLL et de fichiers sources en Visual Basic. 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 Basic gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .vb du répertoire `Sources` pour créer un projet gérant des modules Yoctopuce.

#### Configuration d'un projet Visual Basic

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.vb` 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-basic-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 VisualBasic .NET qui utilise la fonction Led.

```
[...]
Dim errmsg As String
Dim led As YLed

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

REM Pour gérer le hot-plug, on vérifie que le module est là
If (led.isOnline()) Then
    REM Utiliser led.set_power(), ...
End If
```

Voyons maintenant en détail ce que font ces quelques lignes.

### yRegisterHub

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

### yFindLed

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

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

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

## set\_power

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

## Un exemple réel

Lancez Microsoft VisualBasic 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.

```
Module Module1

    Private Sub Usage()
        Dim execname = System.AppDomain.CurrentDomain.FriendlyName
        Console.WriteLine("Usage:")
        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)
    End
    End Sub

    Sub Main()
        Dim argv() As String = System.Environment.GetCommandLineArgs()
        Dim errmsg As String = ""
        Dim target As String
        Dim led As YLed

        Dim on_off As String

        If argv.Length < 3 Then Usage()

        target = argv(1)
        on_off = argv(2).ToUpper()

        REM Setup the API to use local USB devices
        If (yRegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
            Console.WriteLine("RegisterHub error: " + errmsg)
        End
        End If

        If target = "any" Then
            led = yFirstLed()
            If led Is Nothing Then
                Console.WriteLine("No module connected (check USB cable) ")
            End
            End If

        Else
            led = yFindLed(target + ".led")
        End If

        If (led.isOnline()) Then
            If on_off = "ON" Then led.set_power(Y_POWER_ON) Else led.set_power(Y_POWER_OFF)
        Else
            Console.WriteLine("Module not connected (check identification and USB cable)")
        End If
    End Sub
End Module
```

```
End Module
```

## 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.

```
Imports System.IO
Imports System.Environment

Module Module1

Sub usage()
    Console.WriteLine("usage: demo <serial or logical name> [ON/OFF]")
    End
End Sub

Sub Main()
    Dim argv() As String = System.Environment.GetCommandLineArgs()
    Dim errmsg As String = ""
    Dim m As ymodule

    If (yRegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
        Console.WriteLine("RegisterHub error:" + errmsg)
        End
    End If

    If argv.Length < 2 Then usage()

    m = yFindModule(argv(1)) REM use serial or logical name

    If (m.isOnline()) Then
        If argv.Length > 2 Then
            If argv(2) = "ON" Then m.set_beacon(Y_BEACON_ON)
            If argv(2) = "OFF" Then m.set_beacon(Y_BEACON_OFF)
        End If
        Console.WriteLine("serial: " + m.get_serialNumber())
        Console.WriteLine("logical name: " + m.get_logicalName())
        Console.WriteLine("luminosity: " + Str(m.get_luminosity()))
        Console.WriteLine("beacon: ")
        If (m.get_beacon() = Y_BEACON_ON) Then
            Console.WriteLine("ON")
        Else
            Console.WriteLine("OFF")
        End If
        Console.WriteLine("upTime: " + Str(m.get_upTime() / 1000) + " sec")
        Console.WriteLine("USB current: " + Str(m.get_usbCurrent()) + " mA")
        Console.WriteLine("Logs:")
        Console.WriteLine(m.get_lastLogs())
    Else
        Console.WriteLine(argv(1) + " not connected (check identification and USB cable)")
    End If

    End Sub
End Module
```

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

### Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'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 `saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```
Module Module1

Sub usage()
    Console.WriteLine("usage: demo <serial or logical name> <new logical name>")
    End
End Sub

Sub Main()
    Dim argv() As String = System.Environment.GetCommandLineArgs()
    Dim errmsg As String = ""
    Dim newname As String
    Dim m As YModule

    If (argv.Length <> 3) Then usage()

    REM Setup the API to use local USB devices
    If yRegisterHub("usb", errmsg) <> YAPI_SUCCESS Then
        Console.WriteLine("RegisterHub error: " + errmsg)
        End
    End If

    m = yFindModule(argv(1)) REM use serial or logical name
    If m.isOnline() Then

        newname = argv(2)
        If (Not yCheckLogicalName(newname)) Then
            Console.WriteLine("Invalid name (" + newname + ")")
            End
        End If
        m.set_logicalName(newname)
        m.saveToFlash() REM do not forget this

        Console.Write("Module: serial= " + m.get_serialNumber)
        Console.Write(" / name= " + m.get_logicalName())
    Else
        Console.Write("not connected (check identification and USB cable")
    End If

    End Sub
End Module
```

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

## Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `nextModule()` de cet objet pour trouver les modules suivants, et ce tant que la réponse n'est pas un `Nothing`. Ci-dessous un petit exemple listant les module connectés

```
Module Module1

Sub Main()
    Dim M As ymodule
    Dim errmsg As String = ""

    REM Setup the API to use local USB devices
    If yRegisterHub("usb", errmsg) <> YAPI_SUCCESS Then
```

```

    Console.WriteLine("RegisterHub error: " + errmsg)
End
End If

Console.WriteLine("Device list")
M = yFirstModule()
While M IsNot Nothing
    Console.WriteLine(M.get_serialNumber() + " (" + M.get_productName() + ")")
    M = M.nextModule()
End While

End Sub

End Module

```

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

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



### **3. Reference**

## 3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### `yCheckLogicalName(name)`

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

#### `yDisableExceptions()`

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

#### `yEnableExceptions()`

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

#### `yEnableUSBHost(osContext)`

Cette fonction est utilisée uniquement sous Android.

#### `yFreeAPI()`

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

#### `yGetAPIVersion()`

Retourne la version de la librairie Yoctopuce utilisée.

#### `yGetTickCount()`

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

#### `yHandleEvents(errmsg)`

Maintient la communication de la librairie avec les modules Yoctopuce.

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

Initialise la librairie de programmation de Yoctopuce explicitement.

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

Alternative plus tolérante à `RegisterHub()`.

#### `yRegisterDeviceArrivalCallback(arrivalCallback)`

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

#### `yRegisterDeviceRemovalCallback(removalCallback)`

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

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

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

#### `yRegisterHubDiscoveryCallback(hubDiscoveryCallback)`

### 3. Reference

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

#### yRegisterLogFunction(logfun)

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

#### ySelectArchitecture(arch)

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

#### ySetDelegate(object)

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

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

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

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

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

#### yTriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

#### yUnregisterHub(url)

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

#### yUpdateDeviceList(errmsg)

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

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

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

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

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

```
function yCheckLogicalName( ByVal name As String) As Boolean
```

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

**Paramètres :**

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

**Retourne :**

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

## **YAPI.DisableExceptions()**

**YAPI**

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

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

**procedure yDisableExceptions( )**

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

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

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

```
procedure yEnableExceptions( )
```

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

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

---

**YAPI**

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

**procedure yFreeAPI( )**

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

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

Retourne la version de la librairie Yoctopuce utilisée.

```
function yGetAPIVersion( ) As String
```

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

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

**Retourne :**

une chaîne de caractères décrivant la version de la librairie.

## **YAPI.GetTickCount() yGetTickCount()yGetTickCount()**

---

**YAPI**

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

```
function yGetTickCount( ) As Long
```

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

**Retourne :**

un long entier contenant la valeur du compteur de millisecondes.

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

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

```
function yHandleEvents( ByRef errmsg As String) As YRETCODE
```

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

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

### Paramètres :

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

### Retourne :

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

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

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

YAPI

Initialise la librairie de programmation de Yoctopuce explicitement.

```
function yInitAPI( ByVal mode As Integer, ByRef errmsg As String) As Integer
```

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

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

### Paramètres :

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

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

### Retourne :

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

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

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

Alternative plus tolérante à RegisterHub().

```
function yPreregisterHub( ByVal url As String,  
                                ByRef errmsg As String) As Integer
```

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

```
procedure yRegisterDeviceArrivalCallback( ByVal arrivalCallback As yDeviceUpdateFunc)
```

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

**Paramètres :**

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

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

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

```
procedure yRegisterDeviceRemovalCallback( ByVal removalCallback As yDeviceUpdateFunc)
```

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

**Paramètres :**

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

## YAPI.RegisterHub() yRegisterHub()yRegisterHub()

YAPI

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

```
function yRegisterHub( ByVal url As String,  
                      ByRef errmsg As String) As Integer
```

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

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

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

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

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

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

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

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

### Paramètres :

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

### Retourne :

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

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

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

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

```
procedure yRegisterHubDiscoveryCallback( ByVal hubDiscoveryCallback As YHubDiscoveryCallback)
```

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

**Paramètres :**

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

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

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

```
procedure yRegisterLogFunction( ByVal logfun As yLogFunc)
```

Utile pour débugger le fonctionnement de l'API.

**Paramètres :**

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

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

YAPI

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

```
function ySleep( ByVal ms_duration As Integer,  
                                ByRef errmsg As String) As Integer
```

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

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

### Paramètres :

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

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

### Retourne :

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

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

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

Relance une détection des hubs réseau.

```
function yTriggerHubDiscovery( ByRef errmsg As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

```
procedure yUnregisterHub( ByVal url As String)
```

**Paramètres :**

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

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

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

```
function yUpdateDeviceList( ByRef errmsg As String) As YRETCODE
```

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

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

**Paramètres :**

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

**Retourne :**

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

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

## 3.2. Interface de la fonction Accelerometer

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

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

```

### Fonction globales

#### **yFindAccelerometer(func)**

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

#### **yFirstAccelerometer()**

Commence l'énumération des accéléromètres accessibles par la librairie.

### Méthodes des objets YAccelerometer

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

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

#### **accelerometer→describe()**

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

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

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

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

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en g, sous forme de nombre à virgule.

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

Retourne la valeur actuelle de l'accélération, en g, sous forme de nombre à virgule.

#### **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()**

### 3. Reference

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

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

**accelerometer→registerValueCallback(callback)**

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

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

Modifie la mémoire de valeur maximale observée.

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

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

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

Modifie le nom logique de l'accéléromètre.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

**accelerometer→set\_userData(data)**

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

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

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

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

YAccelerometer

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

```
function yFindAccelerometer( ByVal func As String) As YAccelerometer
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

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

### Retourne :

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

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

Commence l'énumération des accéléromètres accessibles par la librairie.

```
function yFirstAccelerometer( ) As YAccelerometer
```

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

**Retourne :**

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

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

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

**procedure calibrateFromPoints( )**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

function **describe( )** As String

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

**Retourne :**

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

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

**YAccelerometer**

---

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

**function get\_advertisedValue( ) As String**

**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), en g, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en g, sous forme de nombre à virgule

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, en g, sous forme de nombre à virgule.

**function get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'accélération, en g, sous forme de nombre à virgule

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.

**function getErrorMessage( ) As String**

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

**Retourne :**

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

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

**YAccelerometer**

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

```
function get_errorType( ) As YRETCODE
```

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

**YAccelerometer**

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

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

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.

**function get\_functionId( ) As String**

Par example `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'accéléromètre (ex: `relay1`)

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

**accelerometer→get.hardwareId()**  
**accelerometer→hardwareId()**  
**accelerometer.get.hardwareId()**

**YAccelerometer**

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

**function get.hardwareId( ) As String**

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

**Retourne :**

une chaîne de caractères identifiant l'accéléromètre (ex: RELAYL01-123456.relay1)

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

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

**YAccelerometer**

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

**function get\_highestValue( ) As Double**

**Retourne :**

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

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

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

**YAccelerometer**

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

function **get\_logFrequency( ) As String**

**Retourne :**

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

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

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

---

**YAccelerometer**

Retourne le nom logique de l'accéléromètre.

```
function get_logicalName( ) As String
```

**Retourne :**

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

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

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

**YAccelerometer**

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

function **get\_lowestValue( ) As Double**

**Retourne :**

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

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

**accelerometer→get\_module()**

**YAccelerometer**

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

---

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

```
function get_module( ) As YModule
```

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

**Retourne :**

une instance de **YModule**

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

**YAccelerometer**

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

**function get\_recordedData( ) As YDataSet**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

**YAccelerometer**

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

function **get\_reportFrequency( ) As String**

**Retourne :**

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

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

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

**YAccelerometer**

Retourne la résolution des valeurs mesurées.

**function get\_resolution( ) As Double**

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

**accelerometer→get\_unit()**

**YAccelerometer**

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

---

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

```
function get_unit( ) As String
```

**Retourne :**

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

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

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

**YAccelerometer**

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

```
function get(userData) As Object
```

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

**accelerometer→get\_xValue()**

**YAccelerometer**

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

---

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

**function get\_xValue( ) As Double**

**Retourne :**

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

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

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

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

```
function get_yValue( ) As Double
```

**Retourne :**

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

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

**accelerometer→get\_zValue()**

**YAccelerometer**

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

---

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

**function get\_zValue( ) As Double**

**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()****YAccelerometer**

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

**function isOnline( ) As Boolean**

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

**Retourne :**

true si l'accéléromètre est joignable, false sinon

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

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

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

procedure **loadCalibrationPoints( )**

**Paramètres :**

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

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

**Retourne :**

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

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

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

**YAccelerometer**

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

```
function nextAccelerometer( ) As YAccelerometer
```

**Retourne :**

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

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

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

```
function registerTimedReportCallback( ) As Integer
```

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.

```
function registerValueCallback( ) As Integer
```

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

**Paramètres :**

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

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

YAccelerometer

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

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

```
function set_logFrequency( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

Modifie le nom logique de l'accéléromètre.

```
function set_logicalName( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

YAccelerometer

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAccelerometer**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

procedure **set(userData)** ByVal **data** As Object

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

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.3. Interface de la fonction Altitude

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_altitude.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAltitude = yoctolib.YAltitude;
php require_once('yocto_altitude.php');
cpp #include "yocto_altitude.h"
m #import "yocto_altitude.h"
pas uses yocto_altitude;
vb yocto_altitude.vb
cs yocto_altitude.cs
java import com.yoctopuce.YoctoAPI.YAltitude;
py from yocto_altitude import *

```

### Fonction globales

#### yFindAltitude(func)

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

#### yFirstAltitude()

Commence l'énumération des altimètres accessibles par la librairie.

### Méthodes des objets YAltitude

#### altitude→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.

#### altitude→describe()

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

#### altitude→get\_advertisedValue()

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

#### altitude→get\_currentRawValue()

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en mètres, sous forme de nombre à virgule.

#### altitude→get\_currentValue()

Retourne la valeur actuelle de l'altitude, en mètres, sous forme de nombre à virgule.

#### altitude→get\_errorMessage()

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

#### altitude→get\_errorType()

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

#### altitude→get\_friendlyName()

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

#### altitude→get\_functionDescriptor()

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

#### altitude→get\_functionId()

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

#### altitude→get\_hardwareId()

Retourne l'identifiant matériel unique de l'altimètre au format SERIAL.FUNCTIONID.
<b>altitude→get_highestValue()</b>
Retourne la valeur maximale observée pour l'altitude depuis le démarrage du module.
<b>altitude→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>altitude→get_logicalName()</b>
Retourne le nom logique de l'altimètre.
<b>altitude→get_lowestValue()</b>
Retourne la valeur minimale observée pour l'altitude depuis le démarrage du module.
<b>altitude→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>altitude→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>altitude→get_qnh()</b>
Retourne la pression de référence au niveau de la mer utilisée pour le calcul de l'altitude (QNH).
<b>altitude→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>altitude→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>altitude→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>altitude→get_unit()</b>
Retourne l'unité dans laquelle l'altitude est exprimée.
<b>altitude→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>altitude→isOnline()</b>
Vérifie si le module hébergeant l'altimètre est joignable, sans déclencher d'erreur.
<b>altitude→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant l'altimètre est joignable, sans déclencher d'erreur.
<b>altitude→load(msValidity)</b>
Met en cache les valeurs courantes de l'altimètre, avec une durée de validité spécifiée.
<b>altitude→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>altitude→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes de l'altimètre, avec une durée de validité spécifiée.
<b>altitude→nextAltitude()</b>
Continue l'énumération des altimètres commencée à l'aide de yFirstAltitude( ).
<b>altitude→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>altitude→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>altitude→set_currentValue(newval)</b>

### 3. Reference

---

Modifie l'altitude actuelle supposée.

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

Modifie la mémoire de valeur maximale observée.

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

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

**altitude→set\_logicalName(newval)**

Modifie le nom logique de l'altimètre.

**altitude→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**altitude→set\_qnh(newval)**

Modifie la pression de référence au niveau de la mer utilisée pour le calcul de l'altitude (QNH).

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**altitude→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**altitude→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).

**altitude→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.

## YAltitude.FindAltitude()

## YAltitude

### yFindAltitude()yFindAltitude()

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

```
function yFindAltitude( ByVal func As String) As YAltitude
```

L'identifiant peut être spécifié sous plusieurs formes:

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

Cette fonction n'exige pas que l'altimètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAltitude.isOnLine()` pour tester si l'altimètre 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'altimètre sans ambiguïté

#### Retourne :

un objet de classe `YAltitude` qui permet ensuite de contrôler l'altimètre.

## **YAltitude.FirstAltitude() yFirstAltitude()yFirstAltitude()**

**YAltitude**

Commence l'énumération des altimètres accessibles par la librairie.

```
function yFirstAltitude( ) As YAltitude
```

Utiliser la fonction `YAltitude.nextAltitude( )` pour itérer sur les autres altimètres.

**Retourne :**

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

**altitude→calibrateFromPoints()**  
**altitude.calibrateFromPoints()****YAltitude**

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

**procedure calibrateFromPoints( )**

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.

**altitude→describe()altitude.describe()****YAltitude**

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

```
function describe( ) As String
```

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

**Retourne :**

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

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

**YAltitude**

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

```
function get_advertisedValue( ) As String
```

**Retourne :**

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

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

<b>altitude→get_currentRawValue()</b>	<b>YAltitude</b>
<b>altitude→currentRawValue()</b>	
<b>altitude.get_currentRawValue()</b>	

---

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en mètres, sous forme de nombre à virgule.

```
function get_currentRawValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en mètres, sous forme de nombre à virgule

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

---

<b>altitude→get_currentValue()</b>	<b>YAltitude</b>
<b>altitude→currentValue()altitude.get_currentValue()</b>	

---

Retourne la valeur actuelle de l'altitude, en mètres, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'altitude, en mètres, sous forme de nombre à virgule

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

**altitude→get\_errorMessage()**

**YAltitude**

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

---

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

**function get\_errorMessage( ) As String**

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'altimètre.

**altitude→get\_errorType()****YAltitude****altitude→errorType()altitude.get\_errorType()**

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

```
function get_errorType( ) As YRETCODE
```

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'altimètre.

`altitude->get_functionDescriptor()`  
`altitude->functionDescriptor()`  
`altitude.get_functionDescriptor()`

**YAltitude**

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

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

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

**altitude→get\_functionId()****YAltitude****altitude→functionId()altitude.get\_functionId()**

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

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'altimètre (ex: `relay1`)

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

**altitude→get\_hardwareId()**

**YAltitude**

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

---

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

```
function get_hardwareId( ) As String
```

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

**Retourne :**

une chaîne de caractères identifiant l'altimètre (ex: RELAYL01-123456.relay1)

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

---

<b>altitude→get_highestValue()</b>	<b>YAltitude</b>
<b>altitude→highestValue()altitude.get_highestValue()</b>	

---

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

```
function get_highestValue( ) As Double
```

**Retourne :**

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

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

**altitude→get\_logFrequency()**

**YAltitude**

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

---

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

**function get\_logFrequency( ) As String**

**Retourne :**

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

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

---

<b>altitude→get_logicalName()</b>	<b>YAltitude</b>
<b>altitude→logicalName()altitude.get_logicalName()</b>	

---

Retourne le nom logique de l'altimètre.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'altimètre.

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

**altitude→get\_lowestValue()**

**YAltitude**

**altitude→lowestValue()altitude.get\_lowestValue()**

---

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

```
function get_lowestValue( ) As Double
```

**Retourne :**

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

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

**altitude→get\_module()****YAltitude****altitude→module()altitude.get\_module()**

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

```
function get_module( ) As YModule
```

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

**Retourne :**

une instance de `YModule`

**altitude→get\_qnh()**

**YAltitude**

**altitude→qnh()|altitude.get\_qnh()**

Retourne la pression de référence au niveau de la mer utilisée pour le calcul de l'altitude (QNH).

**function get\_qnh( ) As Double**

**Retourne :**

une valeur numérique représentant la pression de référence au niveau de la mer utilisée pour le calcul de l'altitude (QNH)

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

**altitude→get\_recordedData()****YAltitude****altitude→recordedData()|altitude.get\_recordedData()**

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

```
function get_recordedData( ) As YDataSet
```

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>altitude→get_reportFrequency()</b>	<b>YAltitude</b>
<b>altitude→reportFrequency()</b>	
<b>altitude.get_reportFrequency()</b>	

---

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

```
function get_reportFrequency( ) As String
```

**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.

**altitude→get\_resolution()****YAltitude****altitude→resolution()altitude.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

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.

**altitude→get\_unit()**

**YAltitude**

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

---

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

```
function get_unit( ) As String
```

**Retourne :**

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

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

**altitude→get(userData)****YAltitude****altitude→userData()altitude.get(userData)**

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

```
function get(userData) As Object
```

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.

**altitude→isOnline()altitude.isOnline()****YAltitude**

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

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de l'altimètre sont valides au moment de l'appel, le module est considéré joignable. 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'altimètre est joignable, false sinon

**altitude→load()altitude.load()****YAltitude**

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

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

**altitude→loadCalibrationPoints()****YAltitude****altitude.loadCalibrationPoints()**

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

procedure **loadCalibrationPoints( )**

**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.

**altitude→nextAltitude()altitude.nextAltitude()****YAltitude**

Continue l'énumération des altimètres commencée à l'aide de `yFirstAltitude()`.

```
function nextAltitude( ) As YAltitude
```

**Retourne :**

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

**altitude→registerTimedReportCallback()  
altitude.registerTimedReportCallback()****YAltitude**

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

```
function registerTimedReportCallback( ) As Integer
```

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.

**altitude→registerValueCallback()**  
**altitude.registerValueCallback()****YAltitude**

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

```
function registerValueCallback( ) As Integer
```

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

**Paramètres :**

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

<b>altitude→set_currentValue()</b>	<b>YAltitude</b>
<b>altitude→setCurrentValue()</b>	
<b>altitude.set_currentValue()</b>	

---

Modifie l'altitude actuelle supposée.

```
function set_currentValue( ByVal newval As Double) As Integer
```

Ceci permet de compenser les changements de pression ou de travailler en mode relatif.

**Paramètres :**

**newval** une valeur numérique représentant l'altitude actuelle supposée

**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>altitude→set_highestValue()</b>	<b>YAltitude</b>
<b>altitude→setHighestValue()</b>	
<b>altitude.set_highestValue()</b>	

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

<b>altitude→set_logFrequency()</b>	<b>YAltitude</b>
<b>altitude→setLogFrequency()</b>	
<b>altitude.set_logFrequency()</b>	

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

```
function set_logFrequency( ByVal newval As String) As Integer
```

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.

**altitude→set\_logicalName()****YAltitude****altitude→setLogicalName()altitude.set\_logicalName()**

Modifie le nom logique de l'altimètre.

```
function set_logicalName( ByVal newval As String) As Integer
```

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'altimètre.

**Retourne :**

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

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

**altitude→set\_lowestValue()**

**YAltitude**

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

---

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**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.

**altitude→set\_qnh()****YAltitude****altitude→setQnh()altitude.set\_qnh()**

Modifie la pression de référence au niveau de la mer utilisée pour le calcul de l'altitude (QNH).

```
function set_qnh( ByVal newval As Double) As Integer
```

Ceci permet de compenser les changements de pression atmosphérique dus au climat.

**Paramètres :**

**newval** une valeur numérique représentant la pression de référence au niveau de la mer utilisée pour le calcul de l'altitude (QNH)

**Retourne :**

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

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

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

**YAltitude**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

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.

**altitude→set\_resolution()****YAltitude****altitude→setResolution()altitude.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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.

**altitude→set(userData)**

**YAltitude**

**altitude→setUserData()altitude.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).

```
procedure set(userData( ByVal data As Object)
```

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

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.4. Interface de la fonction 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 un 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() yFindAnButton()yFindAnButton()

YAnButton

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

```
function yFindAnButton( ByVal func As String) As YAnButton
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

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

### Retourne :

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

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

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

```
function yFirstAnButton( ) As YAnButton
```

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

**Retourne :**

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

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

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

```
function describe() As String
```

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

**Retourne :**

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

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

**YAnButton**

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

function **get\_advertisedValue( ) As String**

**Retourne :**

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

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

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

---

**YAnButton**

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

```
function get_analogCalibration( ) As Integer
```

**Retourne :**

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

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

**anbutton→get\_calibratedValue()**  
**anbutton→calibratedValue()**  
**anbutton.get\_calibratedValue()**

**YAnButton**

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

function **get\_calibratedValue( ) As Integer**

**Retourne :**

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

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

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

**YAnButton**

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

**function get\_calibrationMax( ) As Integer**

**Retourne :**

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

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

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

**YAnButton**

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

function **get\_calibrationMin( ) As Integer**

**Retourne :**

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

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

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

**YAnButton**

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

**function get\_errorMessage( ) As String**

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

**Retourne :**

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

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

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

```
function get_errorType( ) As YRETCODE
```

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

---

**YAnButton**

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

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

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.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'entrée analogique (ex: `relay1`)

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

**anbutton→get\_hardwareId()**

**YAnButton**

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

---

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

```
function get_hardwareId( ) As String
```

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

**Retourne :**

une chaîne de caractères identifiant l'entrée analogique (ex: RELAYL01-123456.relay1)

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

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

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

```
function get_isPressed( ) As Integer
```

**Retourne :**

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

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

**anbutton→get\_lastTimePressed()**  
**anbutton→lastTimePressed()**  
**anbutton.get\_lastTimePressed()**

**YAnButton**

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

function **get\_lastTimePressed( ) As Long**

**Retourne :**

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

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

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

**YAnButton**

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

function **get\_lastTimeReleased( ) As Long**

**Retourne :**

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

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

**anbutton→get\_logicalName()**

**YAnButton**

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

---

Retourne le nom logique de l'entrée analogique.

```
function get_logicalName( ) As String
```

**Retourne :**

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

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

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

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

```
function get_module( ) As YModule
```

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

**Retourne :**

une instance de `YModule`

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

---

**YAnButton**

Retourne la valeur du compteur d'impulsions.

```
function get_pulseCounter( ) As Long
```

**Retourne :**

un entier représentant la valeur du compteur d'impulsions

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

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

Retourne le timer du compteur d'impulsions (ms)

```
function get_pulseTimer( ) As Long
```

**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).

**function get\_rawValue( ) As Integer**

**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.

```
function get_sensitivity( ) As Integer
```

**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).

**function get(userData) As Object**

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

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

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

```
function isOnline( ) As Boolean
```

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

**Retourne :**

true si l'entrée analogique est joignable, false sinon

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

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

```
function load( ByVal msValidity As Integer) As YRETCODE
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

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

```
function nextAnButton( ) As YAnButton
```

**Retourne :**

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

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

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

```
function registerValueCallback( ) As Integer
```

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

**Paramètres :**

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

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

réinitialise le compteur d'impulsions et son timer

```
function resetCounter( ) As Integer
```

**Retourne :**

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

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

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

**YAnButton**

Enclenche ou déclenche le procédure de calibration.

```
function set_analogCalibration( ByVal newval As Integer) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

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

function **set\_calibrationMax( ByVal newval As Integer) As Integer**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

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

function **set\_calibrationMin( ByVal newval As Integer) As Integer**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YAnButton**

Modifie le nom logique de l'entrée analogique.

```
function set_logicalName( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

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

```
function set_sensitivity( ByVal newval As Integer) As Integer
```

La sensibilité sert à filtrer les variations autour d'une valeur fixe, mais ne prétermine 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).

```
procedure set(userData( ByVal data As Object)
```

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

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.5. Interface de la fonction CarbonDioxide

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_carbondioxide.js'></script>
nodejs var yoctolib = require('yoctolib');
var YCarbonDioxide = yoctolib.YCarbonDioxide;
php require_once('yocto_carbondioxide.php');
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 renvoyée par le capteur (sans arrondi ni calibration), en ppm (val), sous forme de nombre à virgule.

#### carbondioxide→get\_currentValue()

Retourne la valeur actuelle du taux de CO2, en ppm (val), sous forme de nombre à virgule.

#### 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.
<b>carbondioxide→get_highestValue()</b>
Retourne la valeur maximale observée pour le taux de CO2 depuis le démarrage du module.
<b>carbondioxide→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>carbondioxide→get_logicalName()</b>
Retourne le nom logique du capteur de CO2.
<b>carbondioxide→get_lowestValue()</b>
Retourne la valeur minimale observée pour le taux de CO2 depuis le démarrage du module.
<b>carbondioxide→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>carbondioxide→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>carbondioxide→get_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>carbondioxide→get_reportFrequency()</b>
Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>carbondioxide→get_resolution()</b>
Retourne la résolution des valeurs mesurées.
<b>carbondioxide→get_unit()</b>
Retourne l'unité dans laquelle le taux de CO2 est exprimée.
<b>carbondioxide→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>carbondioxide→isOnline()</b>
Vérifie si le module hébergeant le capteur de CO2 est joignable, sans déclencher d'erreur.
<b>carbondioxide→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le capteur de CO2 est joignable, sans déclencher d'erreur.
<b>carbondioxide→load(msValidity)</b>
Met en cache les valeurs courantes du capteur de CO2, avec une durée de validité spécifiée.
<b>carbondioxide→loadCalibrationPoints(rawValues, refValues)</b>
Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>carbondioxide→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du capteur de CO2, avec une durée de validité spécifiée.
<b>carbondioxide→nextCarbonDioxide()</b>
Continue l'énumération des capteurs de CO2 commencée à l'aide de yFirstCarbonDioxide( ).
<b>carbondioxide→registerTimedReportCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>carbondioxide→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>carbondioxide→set_highestValue(newval)</b>
Modifie la mémoire de valeur maximale observée.
<b>carbondioxide→set_logFrequency(newval)</b>

### 3. Reference

---

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

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

Modifie le nom logique du capteur de CO2.

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

Modifie la mémoire de valeur minimale observée.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

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

Modifie la résolution des valeurs physique mesurées.

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

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

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

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

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

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

```
function yFindCarbonDioxide( ByVal func As String) As YCarbonDioxide
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

**Paramètres :**

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

**Retourne :**

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

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

---

### **YCarbonDioxide**

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

```
function yFirstCarbonDioxide( ) As YCarbonDioxide
```

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

**Retourne :**

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

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

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

**procedure calibrateFromPoints( )**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

```
function describe( ) As String
```

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

**Retourne :**

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

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

**YCarbonDioxide**

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

```
function get_advertisedValue( ) As String
```

**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), en ppm (val), sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en ppm (val), sous forme de nombre à virgule

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

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

**YCarbonDioxide**

Retourne la valeur actuelle du taux de CO<sub>2</sub>, en ppm (val), sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle du taux de CO<sub>2</sub>, en ppm (val), sous forme de nombre à virgule

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

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

**YCarbonDioxide**

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

```
function get_errorMessage( ) As String
```

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

**Retourne :**

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

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

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO<sub>2</sub>.

```
function get_errorType( ) As YRETCODE
```

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 CO<sub>2</sub>.

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

**YCarbonDioxide**

---

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

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

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

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

**YCarbonDioxide**

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

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de CO2 (ex: `relay1`)

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

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

**YCarbonDioxide**

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

**function get\_hardwareId( ) As String**

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

**Retourne :**

une chaîne de caractères identifiant le capteur de CO2 (ex: RELAYL01-123456.relay1)

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

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

**YCarbonDioxide**

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

function **get\_highestValue( ) As Double**

**Retourne :**

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

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

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

**YCarbonDioxide**

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

**function get\_logFrequency( ) As String**

**Retourne :**

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

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

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

**YCarbonDioxide**

Retourne le nom logique du capteur de CO2.

```
function get_logicalName( ) As String
```

**Retourne :**

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

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

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

**YCarbonDioxide**

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

```
function get_lowestValue( ) As Double
```

**Retourne :**

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

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

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

**YCarbonDioxide**

---

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

```
function get_module( ) As YModule
```

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

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

```
function get_recordedData( ) As YDataSet
```

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

```
function get_reportFrequency( ) As String
```

**Retourne :**

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

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

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

---

**YCarbonDioxide**

Retourne la résolution des valeurs mesurées.

**function get\_resolution( ) As Double**

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

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

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

```
function get_unit( ) As String
```

**Retourne :**

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

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

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

**YCarbonDioxide**

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

```
function get(userData) As Object
```

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

**carbondioxide→isOnline()****YCarbonDioxide**

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

**function isOnline( ) As Boolean**

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

**Retourne :**

true si le capteur de CO2 est joignable, false sinon

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

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

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

```
procedure loadCalibrationPoints( )
```

**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.

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

---

**YCarbonDioxide**

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

```
function nextCarbonDioxide( ) As YCarbonDioxide
```

**Retourne :**

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

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

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

```
function registerTimedReportCallback( ) As Integer
```

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

**Paramètres :**

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

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

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

```
function registerValueCallback( ) As Integer
```

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

**Paramètres :**

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

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

**YCarbonDioxide**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

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

**function set\_logFrequency( ByVal newval As String) As Integer**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCarbonDioxide**

Modifie le nom logique du capteur de CO2.

```
function set_logicalName( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

---

**YCarbonDioxide**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

---

**YCarbonDioxide**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

procedure **set(userData)** ByVal **data** As Object

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

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.6. Interface de la fonction 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() yFindColorLed()yFindColorLed()

YColorLed

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

```
function yFindColorLed( ByVal func As String) As YColorLed
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

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

### Retourne :

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

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

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

```
function yFirstColorLed( ) As YColorLed
```

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

**Retourne :**

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

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

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

```
function describe() As String
```

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

**Retourne :**

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

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

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

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de la led RGB (pas plus de 6 caractères).

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

**colorled→getErrorMessage()**  
**colorled→errorMessage()**  
**colorled.getErrorMessage()**

**YColorLed**

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

```
function getErrorMessage( ) As String
```

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

**Retourne :**

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

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

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

```
function get_errorType( ) As YRETCODE
```

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.

---

<b>colorled→get_functionDescriptor()</b>	<b>YColorLed</b>
<b>colorled→functionDescriptor()</b>	
<b>colorled.get_functionDescriptor()</b>	

---

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

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

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.

```
function get_functionId( ) As String
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant la led RGB (ex: relay1)

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

**colorled→get.hardwareId()**

**YColorLed**

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

---

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

```
function get.hardwareId( ) As String
```

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

**Retourne :**

une chaîne de caractères identifiant la led RGB (ex: RELAYL01-123456.relay1)

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

**colorled→get\_hslColor()****YColorLed****colorled→hslColor()colorled.get\_hslColor()**

Retourne la couleur HSL courante de la led.

```
function get_hslColor( ) As Integer
```

**Retourne :**

un entier représentant la couleur HSL courante de la led

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

**colorled→get\_logicalName()**

**YColorLed**

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

---

Retourne le nom logique de la led RGB.

```
function get_logicalName( ) As String
```

**Retourne :**

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

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

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

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

```
function get_module( ) As YModule
```

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

**Retourne :**

une instance de YModule

**colorled→get\_rgbColor()**

**YColorLed**

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

---

Retourne la couleur RGB courante de la led.

```
function get_rgbColor( ) As Integer
```

**Retourne :**

un entier représentant la couleur RGB courante de la led

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

**colorled→get\_rgbColorAtPowerOn()****YColorLed****colorled→rgbColorAtPowerOn()****colorled.get\_rgbColorAtPowerOn()**

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

```
function get_rgbColorAtPowerOn( ) As Integer
```

**Retourne :**

un entier représentant la couleur configurée pour être affichage à l'allumage du module

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

**colorled→get(userData)**

**YColorLed**

**colorled→userData()colorled.get(userData)**

---

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

```
function get(userData) As Object
```

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

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

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

```
function hsIMove( ByVal hsl_target As Integer,  
                  ByVal ms_duration As Integer) As Integer
```

**Paramètres :**

**hsl\_target** couleur HSL désirée à la fin de la transition  
**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

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

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

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

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

```
function isOnline( ) As Boolean
```

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

**Retourne :**

true si la led RGB est joignable, false sinon

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

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

```
function load( ByVal msValidity As Integer) As YRETCODE
```

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

**Paramètres :**

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

**Retourne :**

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

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

## colorled→nextColorLed()colorled.nextColorLed()

YColorLed

---

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

```
function nextColorLed( ) As YColorLed
```

**Retourne :**

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

**colorled→registerValueCallback()  
colorled.registerValueCallback()****YColorLed**

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

```
function registerValueCallback( ) As Integer
```

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

**Paramètres :**

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

**colorled→rgbMove()colorled.rgbMove()****YColorLed**

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

```
function rgbMove( ByVal rgb_target As Integer,  
                  ByVal ms_duration As Integer) As Integer
```

**Paramètres :**

**rgb\_target** couleur RGB désirée à la fin de la transition  
**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

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

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

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

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

```
function set_hslColor( ByVal newval As Integer) As Integer
```

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

**Paramètres :**

**newval** un entier représentant la couleur courante de la led, en utilisant une couleur HSL spécifiée

**Retourne :**

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

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

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

**YColorLed**

Modifie le nom logique de la led RGB.

```
function set_logicalName( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

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

```
function set_rgbColor( ByVal newval As Integer) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YColorLed**

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

**function set\_rgbColorAtPowerOn( ByVal newval As Integer) As Integer**

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).

```
procedure set(userData( ByVal data As Object)
```

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

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.7. Interface de la fonction 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), en degrés, sous forme de nombre à virgule.

#### compass→get\_currentValue()

Retourne la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule.

#### 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.
<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>

### 3. Reference

---

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)**

Enregistre 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()yFindCompass()**

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

```
function yFindCompass( ByVal func As String) As YCompass
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

**Paramètres :**

**func** une chaîne de caractères qui référence le compas sans ambiguïté

**Retourne :**

un objet de classe `YCompass` qui permet ensuite de contrôler le compas.

## **YCompass.FirstCompass() yFirstCompass()yFirstCompass()**

---

**YCompass**

Commence l'énumération des compas accessibles par la librairie.

```
function yFirstCompass( ) As YCompass
```

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

**Retourne :**

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

**compass→calibrateFromPoints()**  
**compass.calibrateFromPoints()****YCompass**

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

**procedure calibrateFromPoints( )**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

```
function describe( ) As String
```

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

**Retourne :**

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

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

**YCompass**

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

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du compas (pas plus de 6 caractères).

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

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

**YCompass**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule

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, en degrés, sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule

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

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

**YCompass**

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

**function get\_errorMessage( ) As String**

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

**Retourne :**

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

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

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

```
function get_errorType( ) As YRETCODE
```

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

---

**YCompass**

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

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

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

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

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

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

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le compas (ex: `relay1`)

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

**compass→get\_hardwareId()**

**YCompass**

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

---

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

```
function get_hardwareId( ) As String
```

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

**Retourne :**

une chaîne de caractères identifiant le compas (ex: RELAYL01-123456.relay1)

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

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

**YCompass**

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

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le cap relatif depuis le démarrage du module

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

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

**YCompass**

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

function **get\_logFrequency( ) As String**

**Retourne :**

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

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

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

Retourne le nom logique du compas.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du compas.

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

**compass→get\_lowestValue()**

**YCompass**

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

---

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

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le cap relatif depuis le démarrage du module

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

**compass→get\_magneticHeading()**  
**compass→magneticHeading()**  
**compass.get\_magneticHeading()**

**YCompass**

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

```
function get_magneticHeading( ) As Double
```

**Retourne :**

une valeur numérique représentant la direction du nord magnétique, indépendemment du cap configuré

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

**compass→get\_module()**

**YCompass**

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

---

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

```
function get_module( ) As YModule
```

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

**Retourne :**

une instance de YModule

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

**YCompass**

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

**function get\_recordedData( ) As YDataSet**

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

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

**Paramètres :**

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

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

**Retourne :**

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

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

**YCompass**

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

function **get\_reportFrequency( ) As String**

**Retourne :**

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

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

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

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

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

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

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

**compass→get\_unit()**

**YCompass**

**compass→unit()compass.get\_unit()**

---

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

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le cap relatif est exprimée

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

**compass→get(userData)****YCompass****compass→userData()compass.get(userData())**

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

```
function get(userData) As Object
```

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

**Retourne :**

l'objet stocké précédemment par l'appelant.

**compass→isOnline()compass.isOnline()****YCompass**

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

```
function isOnline( ) As Boolean
```

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

**Retourne :**

true si le compas est joignable, false sinon

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

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

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

**procedure loadCalibrationPoints( )**

**Paramètres :**

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

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

**Retourne :**

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

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

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

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

```
function nextCompass( ) As YCompass
```

**Retourne :**

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

**compass→registerTimedReportCallback()  
compass.registerTimedReportCallback()****YCompass**

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

```
function registerTimedReportCallback( ) As Integer
```

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

**Paramètres :**

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

**compass→registerValueCallback()**  
**compass.registerValueCallback()****YCompass**

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

```
function registerValueCallback( ) As Integer
```

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

**Paramètres :**

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

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

---

**YCompass**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

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

**function set\_logFrequency( ByVal newval As String) As Integer**

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie le nom logique du compas.

```
function set_logicalName( ByVal newval As String) As Integer
```

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

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

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

**Retourne :**

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

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

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

**YCompass**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

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

**Paramètres :**

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

**Retourne :**

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

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

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

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

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

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.8. Interface de la fonction 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 renvoyée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule.

#### current→get\_currentValue()

Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.

#### 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 depuis le démarrage du module.
<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 depuis le démarrage du module.
<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.
<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.

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

Modifie la fréquence de notification périodique des valeurs mesurées.

**current→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**current→set\_userData(data)**

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

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

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

## YCurrent.FindCurrent() yFindCurrent()yFindCurrent()

**YCurrent**

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

```
function yFindCurrent( ByVal func As String) As YCurrent
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

### Paramètres :

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

### Retourne :

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

**YCurrent.FirstCurrent()****YCurrent****yFirstCurrent()yFirstCurrent()**

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

```
function yFirstCurrent( ) As YCurrent
```

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

**Retourne :**

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

**current→calibrateFromPoints()**  
**current.calibrateFromPoints()****YCurrent**

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

**procedure calibrateFromPoints( )**

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

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

**Paramètres :**

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

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

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→describe()current.describe()****YCurrent**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de courant (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**current→get\_advertisedValue()**  
**current→advertisedValue()**  
**current.get\_advertisedValue()**

**YCurrent**

---

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

**function get\_advertisedValue( ) As String**

**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\_ADVISEDVALUE\_INVALID.

**current→get\_currentRawValue()**  
**current→currentRawValue()**  
**current.get\_currentRawValue()**

**YCurrent**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

**current→get\_currentValue()**

**YCurrent**

**current→currentValue()current.get\_currentValue()**

---

Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle du courant, en mA, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**current→getErrorMessage()****YCurrent****current→errorMessage()current.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

**current→get\_errorType()**

**YCurrent**

**current→errorType()current.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**current→functionDescriptor()**  
**current.get\_functionDescriptor()**

**YCurrent**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

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.

function **get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de courant (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**current→get\_hardwareId()****YCurrent****current→hardwareId()current.get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de courant au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

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 depuis le démarrage du module.

**function get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le courant depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**current→get\_logFrequency()****YCurrent****current→logFrequency()current.get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( ) As String
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**current→get\_logicalName()**

**YCurrent**

**current→logicalName()current.get\_logicalName()**

---

Retourne le nom logique du capteur de courant.

```
function get_logicalName( ) As String
```

**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 depuis le démarrage du module.

```
function get_lowestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le courant depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**current→get\_module()**

**YCurrent**

**current→module()current.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**current→get\_recordedData()****YCurrent****current→recordedData()current.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( ) As YDataSet
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**current→get\_reportFrequency()**  
**current→reportFrequency()**  
**current.get\_reportFrequency()**

**YCurrent**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**current→get\_resolution()****YCurrent****current→resolution()current.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**current→get\_unit()**

**YCurrent**

**current→unit()current.get\_unit()**

---

Retourne l'unité dans laquelle le courant est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le courant est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**current→get(userData)****YCurrent****current→userData()current.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**current→isOnline()current.isOnline()****YCurrent**

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de courant est joignable, false sinon

**current→load()current.load()****YCurrent**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

procedure **loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→nextCurrent()current.nextCurrent()****YCurrent**

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

```
function nextCurrent( ) As YCurrent
```

**Retourne :**

un pointeur sur un objet `YCurrent` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**current→registerTimedReportCallback()  
current.registerTimedReportCallback()****YCurrent**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**current→set\_highestValue()**  
**current→setHighestValue()**  
**current.set\_highestValue()**

**YCurrent**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_logFrequency()**  
**current→setLogFrequency()**  
**current.set\_logFrequency()**

**YCurrent**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

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.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**current→set\_reportFrequency()**  
**current→setReportFrequency()**  
**current.set\_reportFrequency()**

**YCurrent**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

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 physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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.

**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).

**procedure set(userData( ByVal data As Object)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.9. Interface de la fonction DataLogger

Les capteurs de Yoctopuce sont équipés d'une mémoire non-volatile permettant de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La fonction DataLogger contrôle les paramètres globaux de cet enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_datalogger.js'></script>
node.js var yoctolib = require('yoctolib');
          var YDataLogger = yoctolib.YDataLogger;
php require_once('yocto_datalogger.php');
cpp #include "yocto_datalogger.h"
m #import "yocto_datalogger.h"
pas uses yocto_datalogger;
vb yocto_datalogger.vb
cs yocto_datalogger.cs
java import com.yoctopuce.YoctoAPI.YDataLogger;
py from yocto_datalogger import *

```

### Fonction globales

#### **yFindDataLogger(func)**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

#### **yFirstDataLogger()**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

### Méthodes des objets YDataLogger

#### **datalogger→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format TYPE ( NAME )=SERIAL.FUNCTIONID.

#### **datalogger→forgetAllDataStreams()**

Efface tout l'historique des mesures de l'enregistreur de données.

#### **datalogger→get\_advertisedValue()**

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

#### **datalogger→get\_autoStart()**

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

#### **datalogger→get\_beaconDriven()**

Retourne vrai si l'enregistreur de données est synchronisé avec la balise de localisation.

#### **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()**

### 3. Reference

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.
<b>datalogger→get_friendlyName()</b> Retourne un identifiant global de l'enregistreur de données au format NOM_MODULE . NOM_FONCTION.
<b>datalogger→get_functionDescriptor()</b> Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>datalogger→get_functionId()</b> Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.
<b>datalogger→get_hardwareId()</b> Retourne l'identifiant matériel unique de l'enregistreur de données au format SERIAL . FUNCTIONID.
<b>datalogger→get_logicalName()</b> Retourne le nom logique de l'enregistreur de données.
<b>datalogger→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>datalogger→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>datalogger→get_recording()</b> Retourne l'état d'activation de l'enregistreur de données.
<b>datalogger→get_timeUTC()</b> Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.
<b>datalogger→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>datalogger→isOnline()</b> Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.
<b>datalogger→isOnline_async(callback, context)</b> Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.
<b>datalogger→load(msValidity)</b> Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.
<b>datalogger→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.
<b>datalogger→nextDataLogger()</b> Continue l'énumération des enregistreurs de données commencée à l'aide de yFirstDataLogger( ).
<b>datalogger→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>datalogger→set_autoStart(newval)</b> Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.
<b>datalogger→set_beaconDriven(newval)</b> Modifie le mode de synchronisation de l'enregistreur de données .
<b>datalogger→set_logicalName(newval)</b> Modifie le nom logique de l'enregistreur de données.
<b>datalogger→set_recording(newval)</b> Modifie l'état d'activation de l'enregistreur de données.
<b>datalogger→set_timeUTC(newval)</b> Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.
<b>datalogger→set_userData(data)</b>

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**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() yFindDataLogger()yFindDataLogger()

YDataLogger

Permet de retrouver un enregistreur de données d'après un identifiant donné.

```
function yFindDataLogger( ByVal func As String) As YDataLogger
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'enregistreur de données soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnLine()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

### Retourne :

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

## **YDataLogger.FirstDataLogger() yFirstDataLogger()yFirstDataLogger()**

## **YDataLogger**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

```
function yFirstDataLogger( ) As YDataLogger
```

Utiliser la fonction `YDataLogger.nextDataLogger( )` pour itérer sur les autres enregistreurs de données.

### **Retourne :**

un pointeur sur un objet `YDataLogger`, correspondant au premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

**datalogger→describe()datalogger.describe()****YDataLogger**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**function describe( ) As String**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'enregistreur de données (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**datalogger→forgetAllDataStreams()**  
**datalogger.forgetAllDataStreams()****YDataLogger**

Efface tout l'historique des mesures de l'enregistreur de données.

```
function forgetAllDataStreams( ) As Integer
```

Cette méthode remet aussi à zéro le compteur de Runs.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→get\_advertisedValue()**  
**datalogger→advertisedValue()**  
**datalogger.get\_advertisedValue()**

**YDataLogger**

---

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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\_ADVISEDVALUE\_INVALID.

**datalogger→get\_autoStart()**

**YDataLogger**

**datalogger→autoStart()datalogger.get\_autoStart()**

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

function **get\_autoStart( ) As Integer**

**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\_beaconDriven()**  
**datalogger→beaconDriven()**  
**datalogger.get\_beaconDriven()**

**YDataLogger**

Retourne vrais si l'enregistreur de données est synchronisé avec la balise de localisation.

```
function get_beaconDriven( ) As Integer
```

**Retourne :**

soit Y\_BEACONDRAIVEN\_OFF, soit Y\_BEACONDRAIVEN\_ON, selon vrais si l'enregistreur de données est synchronisé avec la balise de localisation

En cas d'erreur, déclenche une exception ou retourne Y\_BEACONDRAIVEN\_INVALID.

**datalogger→get\_currentRunIndex()**  
**datalogger→currentRunIndex()**  
**datalogger.get\_currentRunIndex()**

**YDataLogger**

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

function **get\_currentRunIndex( ) As Integer**

**Retourne :**

un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRUNINDEX\_INVALID**.

**datalogger→get\_dataSets()**

**YDataLogger**

**datalogger→dataSets()datalogger.get\_dataSets()**

---

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

**function get\_dataSets( ) As List**

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).

procedure **get\_dataStreams(** ByVal **v As List)**

L'appelant doit passer par référence un tableau vide pour stocker les objets YDataStream, et la méthode va les remplir avec des objets décrivant les séquences de données disponibles.

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

**Paramètres :**

**v** un tableau de YDataStreams qui sera rempli avec les séquences trouvées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→get\_errorMessage()**  
**datalogger→errorMessage()**  
**datalogger.get\_errorMessage()**

**YDataLogger**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

**datalogger→get\_errorType()****YDataLogger****datalogger→errorType()datalogger.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**datalogger→functionDescriptor()**  
**datalogger.get\_functionDescriptor()**

---

**YDataLogger**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**datalogger→get\_functionId()****YDataLogger****datalogger→functionId()datalogger.get\_functionId()**

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**datalogger→get\_hardwareId()**  
**datalogger→hardwareId()**  
**datalogger.get\_hardwareId()**

**YDataLogger**

Retourne l'identifiant matériel unique de l'enregistreur de données au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'enregistreur de données (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'enregistreur de données (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**datalogger→get\_logicalName()**  
**datalogger→logicalName()**  
**datalogger.get\_logicalName()**

**YDataLogger**

Retourne le nom logique de l'enregistreur de données.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'enregistreur de données.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**datalogger→get\_module()**

**YDataLogger**

**datalogger→module()datalogger.get\_module()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**datalogger→get\_recording()****YDataLogger****datalogger→recording()datalogger.get\_recording()**

Retourne l'état d'activation de l'enregistreur de données.

```
function get_recording( ) As Integer
```

**Retourne :**

soit Y\_RECORDING\_OFF, soit Y\_RECORDING\_ON, selon l'état d'activation de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_RECORDING\_INVALID.

**datalogger→get\_timeUTC()**

**YDataLogger**

**datalogger→timeUTC()datalogger.get\_timeUTC()**

---

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

```
function get_timeUTC( ) As Long
```

**Retourne :**

un entier représentant le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue

En cas d'erreur, déclenche une exception ou retourne `Y_TIMEUTC_INVALID`.

**datalogger→get(userData)****YDataLogger****datalogger→userData()datalogger.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**datalogger→isOnline()datalogger.isOnline()****YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'enregistreur de données est joignable, false sinon

**datalogger→load()datalogger.load()****YDataLogger**

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→nextDataLogger()**  
**datalogger.nextDataLogger()**

**YDataLogger**

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

```
function nextDataLogger( ) As YDataLogger
```

**Retourne :**

un pointeur sur un objet `YDataLogger` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**datalogger→registerValueCallback()**  
**datalogger.registerValueCallback()****YDataLogger**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**datalogger→set\_autoStart()****YDataLogger****datalogger→setAutoStart()datalogger.set\_autoStart()**

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

```
function set_autoStart( ByVal newval As Integer) As Integer
```

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\_beaconDriven()**  
**datalogger→setBeaconDriven()**  
**datalogger.set\_beaconDriven()**

**YDataLogger**

Modifie le mode de synchronisation de l'enregistreur de données .

```
function set_beaconDriven( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** soit **Y\_BEACONDRIVEN\_OFF**, soit **Y\_BEACONDRIVEN\_ON**, selon le mode de synchronisation 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\_logicalName()**  
**datalogger→setLogicalName()**  
**datalogger.set\_logicalName()**

**YDataLogger**

Modifie le nom logique de l'enregistreur de données.

**function set\_logicalName( ByVal newval As String) As Integer**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'enregistreur de données.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→set\_recording()**  
**datalogger→setRecording()**  
**datalogger.set\_recording()**

**YDataLogger**

Modifie l'état d'activation de l'enregistreur de données.

```
function set_recording( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_RECORDING\_OFF, soit Y\_RECORDING\_ON, selon l'état d'activation de l'enregistreur de données

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→set\_timeUTC()**

**YDataLogger**

**datalogger→setTimeUTC()datalogger.set\_timeUTC()**

---

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

```
function set_timeUTC( ByVal newval As Long) As Integer
```

**Paramètres :**

**newval** un entier représentant la référence de temps UTC, afin de l'attacher aux données enregistrées

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**datalogger→set(userData)****datalogger→setUserData()datalogger.set(userData)****YDataLogger**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
procedure set(userData) ( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.10. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_datalogger.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDataLogger = yoctolib.YDataLogger;
php require_once('yocto_datalogger.php');
cpp #include "yocto_datalogger.h"
m #import "yocto_datalogger.h"
pas uses yocto_datalogger;
vb yocto_datalogger.vb
cs yocto_datalogger.cs
java import com.yoctopuce.YoctoAPI.YDataLogger;
py from yocto_datalogger import *

```

### Méthodes des objets YDataRun

#### **datarun→get\_averageValue(measureName, pos)**

Retourne la valeur moyenne des mesures observées au moment choisi.

#### **datarun→get\_duration()**

Retourne la durée (en secondes) du Run.

#### **datarun→get\_maxValue(measureName, pos)**

Retourne la valeur maximale des mesures observées au moment choisi.

#### **datarun→get\_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

#### **datarun→get\_minValue(measureName, pos)**

Retourne la valeur minimale des mesures observées au moment choisi.

#### **datarun→get\_startTimeUTC()**

Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### **datarun→get\_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

#### **datarun→get\_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

#### **datarun→set\_valueInterval(valueInterval)**

Change l'intervalle de temps représenté par chaque valeur de ce run.

**datarun→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.

## 3.11. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-même sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

Cette classe ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Méthodes des objets YDataSet

#### `dataset→get_endTimeUTC()`

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### `dataset→get_functionId()`

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

#### `dataset→get_hardwareId()`

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

#### `dataset→get_measures()`

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

#### `dataset→get_preview()`

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

#### `dataset→get_progress()`

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

#### `dataset→get_startTimeUTC()`

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

#### `dataset→get_summary()`

Retourne un objet YMeasure résumant tout le YDataSet.

#### `dataset→get_unit()`

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**dataset→loadMore()**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

**dataset→loadMore\_async(callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**dataset→get\_endTimeUTC()****YDataSet****dataset→endTimeUTC()dataset.get\_endTimeUTC()**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

**function get\_endTimeUTC( ) As Long**

Lorsque l'objet YDataSet est créé, l'heure de fin est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore( )`, l'heure de fin est mise à jour à la dernière mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la dernière mesure.

---

**dataset→get\_functionId()****YDataSet****dataset→functionId()dataset.get\_functionId()**

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple temperature1.

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: temperature1)

**dataset→get\_hardwareId()****YDataSet****dataset→hardwareId()dataset.get\_hardwareId()**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple THRMCP1-123456.temperature1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: THRMCP1-123456.temperature1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**dataset→get\_measures()****YDataSet****dataset→measures()dataset.get\_measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

```
function get_measures( ) As List
```

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.

---

<b>dataset→get_preview()</b>	<b>YDataSet</b>
<b>dataset→preview()dataset.get_preview()</b>	

---

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

**function get\_preview( ) As List**

Chaque élément contient: - le début d'un intervalle de temps - la fin d'un intervalle de temps - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Le résumé des mesures est disponible dès que `loadMore( )` a été appelé pour la première fois.

**Retourne :**

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**dataset→get\_progress()****YDataSet****dataset→progress()dataset.get\_progress()**

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

```
function get_progress( ) As Integer
```

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).

**function getStartTimeUTC( ) As Long**

Lorsque l'objet YDataSet est créé, l'heure de départ est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore( )`, l'heure de départ est mise à jour à la première mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la première mesure enregistrée.

---

**dataset→get\_summary()****YDataSet****dataset→summary()dataset.get\_summary()**

---

Retourne un objet YMeasure résumant tout le YDataSet.

```
function get_summary( ) As YMeasure
```

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

**Retourne :**

un objet YMeasure

**dataset→get\_unit()**

**YDataSet**

**dataset→unit()dataset.get\_unit()**

---

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

**function get\_unit( ) As String**

**Retourne :**

une chaîne de caractères représentant une unité physique.

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**dataset→loadMore()dataset.loadMore()****YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

```
function loadMore( ) As Integer
```

**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.12. 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.

**function get\_averageValue( ) As Double**

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la moyenne des valeurs, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→get\_columnCount()**  
**datastream→columnCount()**  
**datastream.get\_columnCount()**

**YDataStream**

Retourne le nombre de colonnes de données contenus dans la séquence.

**function get\_columnCount( ) As Integer**

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames( )`.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclanche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre de colonnes.

En cas d'erreur, déclenche une exception ou retourne zéro.

**datastream→get\_columnNames()**  
**datastream→columnNames()**  
**datastream.get\_columnNames()**

**YDataStream**

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

**function get\_columnNames( ) As List**

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).

```
function get_data( ) As Double
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Paramètres :**

**row** index de l'enregistrement (ligne)

**col** index de la mesure (colonne)

**Retourne :**

un nombre décimal

En cas d'erreur, déclenche une exception ou retourne `Y_DATA_INVALID`.

**datastream→get\_dataRows()****YDataStream****datastream→dataRows()datastream.get\_dataRows()**

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

**function get\_dataRows( ) As List**

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

**Retourne :**

une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**datastream→get\_dataSamplesIntervalMs()**  
**datastream→dataSamplesIntervalMs()**  
**datastream.get\_dataSamplesIntervalMs()**

**YDataStream**

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

**function get\_dataSamplesIntervalMs( ) As Integer**

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la fréquence d'enregistrement peut être changée pour chaque fonction.

**Retourne :**

un entier positif correspondant au nombre de millisecondes entre deux mesures consécutives.

**datastream→get\_duration()**

**YDataStream**

**datastream→duration()datastream.get\_duration()**

---

Retourne la durée approximative de cette séquence, en secondes.

function **get\_duration( ) As Integer**

**Retourne :**

le nombre de secondes couvertes par cette séquence.

En cas d'erreur, déclenche une exception ou retourne Y\_DURATION\_INVALID.

**datastream→get\_maxValue()****YDataStream****datastream→maxValue()datastream.get\_maxValue()**

Retourne la plus grande valeur observée durant cette séquence.

```
function get_maxValue( ) As Double
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus grande valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→get\_minValue()**

**YDataStream**

**datastream→minValue()datastream.get\_minValue()**

---

Retourne la plus petite valeur observée durant cette séquence.

```
function get_minValue( ) As Double
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y\_DATA\_INVALID.

**Retourne :**

un nombre décimal correspondant à la plus petite valeur, ou Y\_DATA\_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y\_DATA\_INVALID.

**datastream→getRowCount()****YDataStream****datastream→rowCount()datastream.getRowCount()**

Retourne le nombre d'enregistrement contenus dans la séquence.

```
function getRowCount( ) As Integer
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclanche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

**Retourne :**

un entier positif correspondant au nombre d'enregistrements.

En cas d'erreur, déclenche une exception ou retourne zéro.

**datastream→get\_runIndex()**

**YDataStream**

**datastream→runIndex()datastream.get\_runIndex()**

---

Retourne le numéro de Run de la séquence de données.

```
function get_runIndex( ) As Integer
```

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

**Retourne :**

un entier positif correspondant au numéro du Run

**datastream→getStartTime()****YDataStream****datastream→startTime()datastream.getStartTime()**

Retourne le temps de départ relatif de la séquence (en secondes).

function **getStartTime( ) As Integer**

Pour les firmwares récents, la valeur est relative à l'heure courante (valeur négative). Pour les modules utilisant un firmware plus ancien que la version 13000, la valeur est le nombre de secondes depuis la mise sous tension du module (valeur positive). Si vous désirez obtenir l'heure absolue du début de la séquence, utilisez `getStartTimeUTC( )`.

**Retourne :**

un entier positif correspondant au nombre de secondes écoulées entre le début du Run et le début de la séquence enregistrée.

**datastream→getStartTimeUTC()**  
**datastream→startTimeUTC()**  
**datastream.getStartTimeUTC()**

**YDataStream**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

**function getStartTimeUTC( ) As Long**

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.

## 3.13. 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()yFindDigitalIO()

YDigitalIO

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

```
function yFindDigitalIO( ByVal func As String) As YDigitalIO
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port d'E/S digital soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDigitalIO.isOnLine()` pour tester si le port d'E/S digital est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le port d'E/S digital sans ambiguïté

### Retourne :

un objet de classe `YDigitalIO` qui permet ensuite de contrôler le port d'E/S digital.

**YDigitalIO.FirstDigitalIO()****yFirstDigitalIO()yFirstDigitalIO()****YDigitalIO**

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

```
function yFirstDigitalIO( ) As YDigitalIO
```

Utiliser la fonction YDigitalIO.nextDigitalIO( ) pour itérer sur les autres ports d'E/S digitaux.

**Retourne :**

un pointeur sur un objet YDigitalIO, correspondant au premier port d'E/S digital accessible en ligne, ou null si il n'y a pas de ports d'E/S digitaux disponibles.

**digitalio→delayedPulse()|digitalio.delayedPulse()****YDigitalIO**

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

```
function delayedPulse( ) As Integer
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**ms\_delay** délai d'attente avant l'impulsion, en millisecondes

**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→describe()digitalio.describe()****YDigitalIO**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le port d'E/S digital (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**digitalio→get\_advertisedValue()**  
**digitalio→advertisedValue()**  
**digitalio.get\_advertisedValue()**

**YDigitalIO**

---

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du port d'E/S digital (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**digitalio→get\_bitDirection()****YDigitalIO****digitalio→bitDirection()digitalio.get\_bitDirection()**

Retourne la direction d'un seul bit du port d'E/S.

```
function get_bitDirection( ) As Integer
```

(0 signifie que le bit est une entrée, 1 une sortie)

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→get\_bitOpenDrain()** YDigitalIO  
**digitalio→bitOpenDrain()digitalio.get\_bitOpenDrain()**

---

Retourne la direction d'un seul bit du port d'E/S.

```
function get_bitOpenDrain( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert)..

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→get\_bitPolarity()****YDigitalIO****digitalio→bitPolarity()digitalio.get\_bitPolarity()**

Retourne la polarité d'un seul bit du port d'E/S.

```
function get_bitPolarity( ) As Integer
```

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.

```
function get_bitState( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

l'état du bit (0 ou 1).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→get\_errorMessage()**  
**digitalio→errorMessage()**  
**digitalio.get\_errorMessage()**

YDigitalIO

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
function get_errorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

**digitalio→get\_errorType()**

**YDigitalIO**

**digitalio→errorType()digitalio.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**digitalio→functionDescriptor()**  
**digitalio.get\_functionDescriptor()****YDigitalIO**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

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.

**function get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**digitalio→get\_hardwareId()****YDigitalIO****digitalio→hardwareId()digitalio.get\_hardwareId()**

Retourne l'identifiant matériel unique du port d'E/S digital au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port d'E/S digital (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le port d'E/S digital (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**digitalio→get\_logicalName()**

**YDigitalIO**

**digitalio→logicalName()digitalio.get\_logicalName()**

---

Retourne le nom logique du port d'E/S digital.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port d'E/S digital.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**digitalio→get\_module()****YDigitalIO****digitalio→module()digitalio.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**digitalio→get\_outputVoltage()**  
**digitalio→outputVoltage()**  
**digitalio.get\_outputVoltage()**

**YDigitalIO**

Retourne la source de tension utilisée pour piloter les bits en sortie.

```
function get_outputVoltage( ) As Integer
```

**Retourne :**

une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUTVOLTAGE\_INVALID.

**digitalio→get\_portDirection()****YDigitalIO****digitalio→portDirection()digitalio.get\_portDirection()**

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
function get_portDirection( ) As Integer
```

**Retourne :**

un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

En cas d'erreur, déclenche une exception ou retourne Y\_PORTDIRECTION\_INVALID.

**digitalio→get\_portOpenDrain()**  
**digitalio→portOpenDrain()**  
**digitalio.get\_portOpenDrain()**

**YDigitalIO**

Retourne le type d'interface électrique de chaque bit du port (bitmap).

```
function get_portOpenDrain( ) As Integer
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

**Retourne :**

un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y\_PORTOPENDRAIN\_INVALID.

**digitalio→get\_portPolarity()****YDigitalIO****digitalio→portPolarity()digitalio.get\_portPolarity()**

Retourne la polarité des bits du port (bitmap).

```
function get_portPolarity( ) As Integer
```

Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

**Retourne :**

un entier représentant la polarité des bits du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y\_PORTPOLARITY\_INVALID.

**digitalio→get\_portSize()**

**YDigitalIO**

**digitalio→portSize()digitalio.get\_portSize()**

---

Retourne le nombre de bits implémentés dans le port d'E/S.

```
function get_portSize( ) As Integer
```

**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\_PORTSIZERO\_INVALID.

**digitalio→get\_portState()****YDigitalIO****digitalio→portState()digitalio.get\_portState()**

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
function get_portState( ) As Integer
```

**Retourne :**

un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

En cas d'erreur, déclenche une exception ou retourne Y\_PORTSTATE\_INVALID.

**digitalio→get(userData)**

**YDigitalIO**

**digitalio→userData()digitalio.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**digitalio→isOnline()digitalio.isOnline()****YDigitalIO**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le port d'E/S digital est joignable, false sinon

**digitalio→load()|digitalio.load()****YDigitalIO**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→nextDigitalIO() digitalio.nextDigitalIO()****YDigitalIO**

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

```
function nextDigitalIO( ) As YDigitalIO
```

**Retourne :**

un pointeur sur un objet `YDigitalIO` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**digitalio→pulse()**digitalio.pulse()****

YDigitalIO

Déclenche une impulsion de durée spécifiée sur un bit choisi.

**function pulse( ) As Integer**

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**ms\_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→registerValueCallback()**  
**digitalio.registerValueCallback()****YDigitalIO**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**digitalio→set\_bitDirection()** YDigitalIO  
**digitalio→setBitDirection()digitalio.set\_bitDirection()**

Change la direction d'un seul bit du port d'E/S.

```
function set_bitDirection( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitdirection** nouvelle valeur de la direction, 0=entrée, 1=sortie. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_bitOpenDrain()**  
**digitalio→setBitOpenDrain()**  
**digitalio.set\_bitOpenDrain()**

YDigitalIO

Change le type d'interface électrique d'un seul bit du port d'E/S.

```
function set_bitOpenDrain( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**opendrain** 0 pour faire une entrée ou une sortie digitale standard, 1 pour une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash( )` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_bitPolarity()****YDigitalIO****digitalio→setBitPolarity()digitalio.set\_bitPolarity()**

Change la polarité d'un seul bit du port d'E/S.

```
function set_bitPolarity( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitpolarity** nouvelle valeur de la polarité. 0=mode normal, 1=mode inverse. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_bitState()****YDigitalIO****digitalio→setBitState()digitalio.set\_bitState()**

Change l'état d'un seul bit du port d'E/S.

```
function set_bitState( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**bitstate** nouvel état du bit (1 ou 0)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_logicalName()**  
**digitalio→setLogicalName()**  
**digitalio.set\_logicalName()**

**YDigitalIO**

Modifie le nom logique du port d'E/S digital.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du port d'E/S digital.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_outputVoltage()  
digitalio→setOutputVoltage()  
digitalio.set\_outputVoltage()****YDigitalIO**

Modifie la source de tension utilisée pour piloter les bits en sortie.

```
function set_outputVoltage( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé après un redémarrage du module.

**Paramètres :**

**newval** une valeur parmi Y\_OUTPUTVOLTAGE\_USB\_5V, Y\_OUTPUTVOLTAGE\_USB\_3V et Y\_OUTPUTVOLTAGE\_EXT\_V représentant la source de tension utilisée pour piloter les bits en sortie

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portDirection()**  
**digitalio→setPortDirection()**  
**digitalio.set\_portDirection()**

**YDigitalIO**

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
function set_portDirection( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portOpenDrain()**  
**digitalio→setPortOpenDrain()**  
**digitalio.set\_portOpenDrain()**

YDigitalIO

Modifie le type d'interface électrique de chaque bit du port (bitmap).

```
function set_portOpenDrain( ByVal newval As Integer) As Integer
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portPolarity()** **YDigitalIO**  
**digitalio→setPortPolarity()digitalio.set\_portPolarity()**

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

```
function set_portPolarity( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**digitalio→set\_portState()****YDigitalIO****digitalio→setPortState()digitalio.set\_portState()**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
function set_portState( ByVal newval As Integer) As Integer
```

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).

```
procedure set(userData)( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**digitalio→toggle\_bitState()digitalio.toggle\_bitState()****YDigitalIO**

Inverse l'état d'un seul bit du port d'E/S.

```
function toggle_bitState( ) As Integer
```

**Paramètres :**

**bitno** index du bit dans le port; le bit de poids faible est à l'index 0

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.14. Interface de la fonction Display

L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_display.js'></script>
nodejs var yoctolib = require('yoctolib');
var YDisplay = yoctolib.YDisplay;
require_once('yocto_display.php');
cpp #include "yocto_display.h"
m #import "yocto_display.h"
pas uses yocto_display;
vb yocto_display.vb
cs yocto_display.cs
java import com.yoctopuce.YoctoAPI.YDisplay;
py from yocto_display import *

```

### Fonction globales

#### yFindDisplay(func)

Permet de retrouver un ecran d'après un identifiant donné.

#### yFirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

### Méthodes des objets YDisplay

#### display→copyLayerContent(srcLayerId, dstLayerId)

Copie le contenu d'un couche d'affichage vers une autre couche.

#### display→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format TYPE(NAME)=SERIAL.FUNCTIONID.

#### display→fade(brightness, duration)

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

#### display→get\_advertisedValue()

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

#### display→get\_brightness()

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

#### display→get\_displayHeight()

Retourne la hauteur de l'écran, en pixels.

#### display→get\_displayLayer(layerId)

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

#### display→get\_displayType()

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

#### display→get\_displayWidth()

Retourne la largeur de l'écran, en pixels.

#### display→get\_enabled()

Retourne vrai si le l'écran est alimenté, faux sinon.

#### display→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

#### **display→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

#### **display→get\_friendlyName()**

Retourne un identifiant global de l'écran au format NOM\_MODULE . NOM\_FONCTION.

#### **display→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **display→get\_functionId()**

Retourne l'identifiant matériel de l'écran, sans référence au module.

#### **display→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'écran au format SERIAL . FUNCTIONID.

#### **display→get\_layerCount()**

Retourne le nombre des couches affichables disponibles.

#### **display→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

#### **display→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

#### **display→get\_logicalName()**

Retourne le nom logique de l'écran.

#### **display→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **display→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **display→get\_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

#### **display→get\_startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

#### **display→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **display→isOnline()**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

#### **display→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

#### **display→load(msValidity)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

#### **display→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

#### **display→newSequence()**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

#### **display→nextDisplay()**

Continue l'énumération des écrans commencée à l'aide de yFirstDisplay( ).

#### **display→pauseSequence(delay\_ms)**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

**display→playSequence(sequenceName)**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes newSequence( ) et saveSequence( ).

**display→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**display→resetAll()**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

**display→saveSequence(sequenceName)**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

**display→set\_brightness(newval)**

Modifie la luminosité de l'écran.

**display→set\_enabled(newval)**

Modifie l'état d'activité de l'écran.

**display→set\_logicalName(newval)**

Modifie le nom logique de l'écran.

**display→set\_orientation(newval)**

Modifie l'orientation de l'écran.

**display→set\_startupSeq(newval)**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

**display→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**display→stopSequence(sequenceName)**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

**display→swapLayerContent(layerIdA, layerIdB)**

Permute le contenu de deux couches d'affichage.

**display→upload(pathname, content)**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

**display→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YDisplay.FindDisplay() yFindDisplay()yFindDisplay()

**YDisplay**

Permet de retrouver un ecran d'après un identifiant donné.

```
function yFindDisplay( ByVal func As String ) As YDisplay
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'écran soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDisplay.isOnLine()` pour tester si l'écran est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'écran sans ambiguïté

**Retourne :**

un objet de classe `YDisplay` qui permet ensuite de contrôler l'écran.

## **YDisplay.FirstDisplay() yFirstDisplay()yFirstDisplay()**

---

**YDisplay**

Commence l'énumération des écran accessibles par la librairie.

```
function yFirstDisplay( ) As YDisplay
```

Utiliser la fonction `YDisplay.nextDisplay( )` pour itérer sur les autres écran.

**Retourne :**

un pointeur sur un objet `YDisplay`, correspondant au premier écran accessible en ligne, ou `null` si il n'y a pas de écran disponibles.

**display→copyLayerContent()  
display.copyLayerContent()****YDisplay**

Copie le contenu d'un couche d'affichage vers une autre couche.

```
function copyLayerContent( ) As Integer
```

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

**Paramètres :**

**srcLayerId** l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

**dstLayerId** l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→describe()display.describe()****YDisplay**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format TYPE ( NAME )=SERIAL.FUNCTIONID.

**function describe( ) As String**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'écran (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

**display→fade()display.fade()****YDisplay**

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

```
function fade( ) As Integer
```

**Paramètres :**

**brightness** nouvelle valeur de luminosité de l'écran

**duration** durée en millisecondes de la transition.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→get\_advertisedValue()**  
**display→advertisedValue()**  
**display.get\_advertisedValue()**

**YDisplay**

---

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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\_ADVISEDVALUE\_INVALID.

**display→get\_brightness()****YDisplay****display→brightness()display.get\_brightness()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
function get_brightness( ) As Integer
```

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne Y\_BRIGHTNESS\_INVALID.

**display→get\_displayHeight()**

**YDisplay**

**display→displayHeight()display.get\_displayHeight()**

---

Retourne la hauteur de l'écran, en pixels.

```
function get_displayHeight( ) As Integer
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne **Y\_DISPLAYHEIGHT\_INVALID**.

**display→get\_displayLayer()****YDisplay****display→displayLayer()display.get\_displayLayer()**

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

```
function get_displayLayer( ) As YDisplayLayer
```

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

**Paramètres :**

**layerId** l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

**Retourne :**

un objet YDisplayLayer

En cas d'erreur, déclenche une exception ou retourne null.

**display→get\_displayType()**

**YDisplay**

**display→displayType()display.get\_displayType()**

---

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

```
function get_displayType( ) As Integer
```

**Retourne :**

une valeur parmi Y\_DISPLAYTYPE\_MONO, Y\_DISPLAYTYPE\_GRAY et Y\_DISPLAYTYPE\_RGB  
représentant le type de l'écran: monochrome, niveaux de gris ou couleur

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYTYPE\_INVALID.

**display→get\_displayWidth()****YDisplay****display→displayWidth()display.get\_displayWidth()**

Retourne la largeur de l'écran, en pixels.

```
function get_displayWidth( ) As Integer
```

**Retourne :**

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYWIDTH\_INVALID.

**display→get\_enabled()** YDisplay  
**display→enabled()display.get\_enabled()**

---

Retourne vrai si le l'écran est alimenté, faux sinon.

```
function get_enabled( ) As Integer
```

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon vrai si le l'écran est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**display→getErrorMessage()****YDisplay****display→errorMessage()display.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

**display→get\_errorType()**

**YDisplay**

**display→errorType()display.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**display→functionDescriptor()**  
**display.get\_functionDescriptor()****YDisplay**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**display→get\_functionId()**

**YDisplay**

**display→functionId()display.get\_functionId()**

---

Retourne l'identifiant matériel de l'écran, sans référence au module.

**function get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'écran (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**display→get\_hardwareId()****YDisplay****display→hardwareId()display.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'écran au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

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.

```
function get_layerCount( ) As Integer
```

**Retourne :**

un entier représentant le nombre des couches affichables disponibles

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERCOUNT_INVALID`.

**display→get\_layerHeight()****YDisplay****display→layerHeight()display.get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

```
function get_layerHeight( ) As Integer
```

**Retourne :**

un entier représentant la hauteur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne **Y\_LAYERHEIGHT\_INVALID**.

**display→get\_layerWidth()**

**YDisplay**

**display→layerWidth()display.get\_layerWidth()**

---

Retourne la largeur des couches affichables, en pixels.

```
function get_layerWidth( ) As Integer
```

**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.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'écran.

En cas d'erreur, déclenche une exception ou retourne **Y\_LOGICALNAME\_INVALID**.

**display→get\_module()** YDisplay  
**display→module()display.get\_module()**

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**display→get\_orientation()****YDisplay****display→orientation()display.get\_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

```
function get_orientation( ) As Integer
```

**Retourne :**

une valeur parmi Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP, Y\_ORIENTATION\_RIGHT et Y\_ORIENTATION\_DOWN représentant l'orientation sélectionnée pour l'écran

En cas d'erreur, déclenche une exception ou retourne Y\_ORIENTATION\_INVALID.

**display→get\_startupSeq()**

**YDisplay**

**display→startupSeq()display.get\_startupSeq()**

---

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

```
function get_startupSeq( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

En cas d'erreur, déclenche une exception ou retourne `Y_STARTUPSEQ_INVALID`.

**display→get(userData)****YDisplay****display→userData()display.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## display→isOnline()display.isOnline()

## YDisplay

---

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de l'écran sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'écran est joignable, false sinon

**display→load()display.load()****YDisplay**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→newSequence()display.newSequence()****YDisplay**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

```
function newSequence( ) As Integer
```

Le nom de la séquence sera donné au moment de l'appel à `saveSequence()`, une fois la séquence terminée.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→nextDisplay()display.nextDisplay()****YDisplay**

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

```
function nextDisplay( ) As YDisplay
```

**Retourne :**

un pointeur sur un objet `YDisplay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**display→pauseSequence()display.pauseSequence()****YDisplay**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

```
function pauseSequence( ) As Integer
```

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

**Paramètres :**

**delay\_ms** la durée de l'attente, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→playSequence()  
display.playSequence()****YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes newSequence( ) et saveSequence( ).

```
function playSequence( ) As Integer
```

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→registerValueCallback()  
display.registerValueCallback()****YDisplay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**display→resetAll()display.resetAll()****YDisplay**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

```
function resetAll( ) As Integer
```

Utiliser cette fonction dans une sequence va tuer stopper l'affichage de la sequence: ne pas utiliser cette fonction pour réinitialiser l'écran au début d'une séquence.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→saveSequence()display.saveSequence()****YDisplay**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

```
function saveSequence( ) As Integer
```

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**display→set\_brightness()  
display→setBrightness()display.set\_brightness()****YDisplay**

Modifie la luminositéde l'écran.

```
function set_brightness( ByVal newval As Integer) As Integer
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** un entier représentant la luminositéde l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_enabled()** YDisplay  
**display→setEnabled()display.set\_enabled()**

---

Modifie l'état d'activité de l'écran.

```
function set_enabled( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état d'activité de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**display→set\_logicalName()** **YDisplay**  
**display→setLogicalName()display.set\_logicalName()**

---

Modifie le nom logique de l'écran.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'écran.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_orientation()  
display→setOrientation()display.set\_orientation()****YDisplay**

Modifie l'orientation de l'écran.

```
function set_orientation( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi Y\_ORIENTATION\_LEFT, Y\_ORIENTATION\_UP,  
Y\_ORIENTATION\_RIGHT et Y\_ORIENTATION\_DOWN représentant l'orientation de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set\_startupSeq()****YDisplay****display→setStartupSeq()display.set\_startupSeq()**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
function set_startupSeq( ByVal newval As String) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→set(userData)** YDisplay

**display→setUserData()display.set(userData)**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**display→stopSequence()display.stopSequence()****YDisplay**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

```
function stopSequence( ) As Integer
```

L'affichage est laissé tel quel.

**Paramètres :**

**sequenceName** le nom de la nouvelle séquence créée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**display→swapLayerContent()  
display.swapLayerContent()****YDisplay**

Permute le contenu de deux couches d'affichage.

```
function swapLayerContent( ) As Integer
```

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.

**procedure upload( )**

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

**Paramètres :**

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.15. 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()**

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 copie d'images bitmaps.

**displaylayer→selectFont(fontname)**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

**displaylayer→selectGrayPen(graylevel)**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

**displaylayer→setAntialiasingMode(mode)**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

**displaylayer→setConsoleBackground(bgcol)**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

**displaylayer→setConsoleMargins(x1, y1, x2, y2)**

Configure les marges d'affichage pour la fonction `consoleOut`.

**displaylayer→setConsoleWordWrap(wordwrap)**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

**displaylayer→setLayerPosition(x, y, scrollTime)**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

**displaylayer→unhide()**

Affiche la couche.

**displaylayer→clear()  
displaylayer.clear()****YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

```
function clear( ) As Integer
```

Cette méthode ne change pas les réglages de la couche. Si vous désirez remettre la couche dans son état initial, utilisez plutôt la méthode `reset()`.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→clearConsole()  
displaylayer.clearConsole()****YDisplayLayer**

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

```
function clearConsole( ) As Integer
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→consoleOut()displaylayer.consoleOut()****YDisplayLayer**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

**function consoleOut( ) As Integer**

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

**Paramètres :**

**text** le message à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBar()displaylayer.drawBar()****YDisplayLayer**

Dessine un rectangle plein à une position spécifiée.

```
function drawBar( ) As Integer
```

**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.

**procedure drawBitmap( )**

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour lasser les pixels inchangés

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawCircle()displaylayer.drawCircle()****YDisplayLayer**

Dessine un cercle vide à une position spécifiée.

```
function drawCircle( ) As Integer
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle

**y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle

**r** le rayon du cercle, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawDisc()displaylayer.drawDisc()****YDisplayLayer**

Dessine un disque plein à une position spécifiée.

```
function drawDisc( ) As Integer
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque

**y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque

**r** le rayon du disque, en pixels

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawImage()displaylayer.drawImage()****YDisplayLayer**

Dessine une image GIF à la position spécifiée de la couche.

**function drawImage( ) As Integer**

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image
- imagename** le nom du fichier GIF à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawPixel()displaylayer.drawPixel()****YDisplayLayer**

Dessine un pixel unique à une position spécifiée.

```
function drawPixel( ) As Integer
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche

**y** la distance en pixels depuis le haut de la couche

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawRect()displaylayer.drawRect()****YDisplayLayer**

Dessine un rectangle vide à une position spécifiée.

```
function drawRect( ) As Integer
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

**y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

**x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

**y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawText()displaylayer.drawText()****YDisplayLayer**

Affiche un texte à la position spécifiée de la couche.

**function drawText( ) As Integer**

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte

**y** la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte

**anchor** le point d'ancrage du texte, choisi parmi l'énumération Y\_ALIGN: Y\_ALIGN\_TOP\_LEFT, Y\_ALIGN\_CENTER\_LEFT, Y\_ALIGN\_BASELINE\_LEFT, Y\_ALIGN\_BOTTOM\_LEFT, Y\_ALIGN\_TOP\_CENTER, Y\_ALIGN\_CENTER, Y\_ALIGN\_BASELINE\_CENTER, Y\_ALIGN\_BOTTOM\_CENTER, Y\_ALIGN\_TOP\_DECIMAL, Y\_ALIGN\_CENTER\_DECIMAL, Y\_ALIGN\_BASELINE\_DECIMAL, Y\_ALIGN\_BOTTOM\_DECIMAL, Y\_ALIGN\_TOP\_RIGHT, Y\_ALIGN\_CENTER\_RIGHT, Y\_ALIGN\_BASELINE\_RIGHT, Y\_ALIGN\_BOTTOM\_RIGHT.

**text** le texte à afficher

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→get\_display()****YDisplayLayer****displaylayer→display()displaylayer.get\_display()**

Retourne l'YDisplay parent.

```
function get_display( ) As YDisplay
```

Retourne l'objet YDisplay parent du YDisplayLayer courant.

**Retourne :**

un objet YDisplay

**displaylayer→get\_displayHeight()**  
**displaylayer→displayHeight()**  
**displaylayer.get\_displayHeight()**

---

**YDisplayLayer**

Retourne la hauteur de l'écran, en pixels.

```
function get_displayHeight( ) As Integer
```

**Retourne :**

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYHEIGHT\_INVALID.

**displaylayer→get\_displayWidth()  
displaylayer→displayWidth()  
displaylayer.get\_displayWidth()****YDisplayLayer**

Retourne la largeur de l'écran, en pixels.

```
function get_displayWidth( ) As Integer
```

**Retourne :**

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_DISPLAYWIDTH\_INVALID.

**displaylayer→get\_layerHeight()**  
**displaylayer→layerHeight()**  
**displaylayer.get\_layerHeight()**

**YDisplayLayer**

Retourne la hauteur des couches affichables, en pixels.

```
function get_layerHeight( ) As Integer
```

**Retourne :**

un entier représentant la hauteur des couches affichables, en pixels. En cas d'erreur, déclenche une exception ou retourne Y\_LAYERHEIGHT\_INVALID.

**displaylayer→get\_layerWidth()**  
**displaylayer→layerWidth()**  
**displaylayer.get\_layerWidth()****YDisplayLayer**

Retourne la largeur des couches affichables, en pixels.

```
function get_layerWidth( ) As Integer
```

**Retourne :**

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y\_LAYERWIDTH\_INVALID.

**displaylayer→hide()displaylayer.hide()****YDisplayLayer**

Cache la couche de dessin.

```
function hide( ) As Integer
```

L'état de la couche est préservé, mais la couche ne sera plus affichée à l'écran jusqu'au prochain appel à `unhide()`. Le fait de cacher la couche améliore les performances de toutes les primitives d'affichage, car il évite de consacrer inutilement des cycles de calcul à afficher les états intermédiaires (technique de double-buffering).

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→lineTo()displaylayer.lineTo()****YDisplayLayer**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

function **lineTo( ) As Integer**

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

**Paramètres :**

- x** la distance en pixels depuis la gauche de la couche jusqu'au point final
- y** la distance en pixels depuis le haut de la couche jusqu'au point final

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→moveTo()displaylayer.moveTo()****YDisplayLayer**

Déplace le point de dessin courant de cette couche à la position spécifiée.

```
function moveTo( ) As Integer
```

**Paramètres :**

**x** la distance en pixels depuis la gauche de la couche de dessin

**y** la distance en pixels depuis le haut de la couche de dessin

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→reset()displaylayer.reset()****YDisplayLayer**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

function **reset( )** As Integer

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear( )`.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectColorPen()**  
**displaylayer.selectColorPen()**

---

**YDisplayLayer**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

```
function selectColorPen( ) As Integer
```

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 copie d'images bitmaps.

```
function selectEraser( ) As Integer
```

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectFont()displaylayer.selectFont()****YDisplayLayer**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

**function selectFont( ) As Integer**

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

**Paramètres :**

**fontname** le nom du fichier définissant la police de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectGrayPen()  
displaylayer.selectGrayPen()****YDisplayLayer**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

function **selectGrayPen( ) As Integer**

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), tout valeur inférieure à 128 conduit à un point noir, et toute valeur supérieure ou égale à 128 devient un point lumineux.

**Paramètres :**

**graylevel** le niveau de gris désiré, de 0 à 255

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setAntialiasingMode()**  
**displaylayer.setAntialiasingMode()****YDisplayLayer**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

**function setAntialiasingMode( ) As Integer**

L'anti-aliasing est atténuée la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

**Paramètres :**

**mode** true pour activer l'antialiasing, false pour le désactiver.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleBackground()**  
**displaylayer.setConsoleBackground()****YDisplayLayer**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

```
function setConsoleBackground( ) As Integer
```

**Paramètres :**

**bgcol** le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleMargins()  
displaylayer.setConsoleMargins()****YDisplayLayer**

Configure les marges d'affichage pour la fonction consoleOut.

```
function setConsoleMargins( ) As Integer
```

**Paramètres :**

**x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche

**y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure

**x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite

**y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setConsoleWordWrap()  
displaylayer.setConsoleWordWrap()****YDisplayLayer**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

```
function setConsoleWordWrap( ) As Integer
```

**Paramètres :**

**wordwrap** true pour retourner à la ligne entre les mots seulement, false pour retourner à l'extrême droite de chaque ligne.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→setLayerPosition()  
displaylayer.setLayerPosition()****YDisplayLayer**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

**function setLayerPosition( ) As Integer**

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

**Paramètres :**

**x** la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.

**y** la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.

**scrollTime** durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→unhide()displaylayer.unhide()****YDisplayLayer**

Affiche la couche.

```
function unhide( ) As Integer
```

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.

## 3.16. 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() yFindDualPower()yFindDualPower()

YDualPower

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

```
function yFindDualPower( ByVal func As String) As YDualPower
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'alimentation soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDualPower.isOnLine()` pour tester si le contrôle d'alimentation est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le contrôle d'alimentation sans ambiguïté

### Retourne :

un objet de classe `YDualPower` qui permet ensuite de contrôler le contrôle d'alimentation.

## YDualPower.FirstDualPower() yFirstDualPower()yFirstDualPower()

## YDualPower

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

```
function yFirstDualPower( ) As YDualPower
```

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

### Retourne :

un pointeur sur un objet `YDualPower`, correspondant au premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

**dualpower→describe()dualpower.describe()****YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le contrôle d'alimentation (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**dualpower→get\_advertisedValue()**  
**dualpower→advertisedValue()**  
**dualpower.get\_advertisedValue()**

**YDualPower**

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

**dualpower→get\_errorType()****YDualPower****dualpower→errorType()dualpower.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
function get_errorType( ) As YRETCODE
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

**dualpower→get\_extVoltage()**

**YDualPower**

**dualpower→extVoltage()dualpower.get\_extVoltage()**

---

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

```
function get_extVoltage( ) As Integer
```

**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\_functionDescriptor()  
dualpower→functionDescriptor()  
dualpower.get\_functionDescriptor()****YDualPower**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

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.

**function get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**dualpower→get\_hardwareId()****YDualPower****dualpower→hardwareId()dualpower.get\_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'alimentation au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'alimentation (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le contrôle d'alimentation (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**dualpower→get\_logicalName()**  
**dualpower→logicalName()**  
**dualpower.get\_logicalName()**

---

**YDualPower**

Retourne le nom logique du contrôle d'alimentation.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**dualpower→get\_module()****YDualPower****dualpower→module()dualpower.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**dualpower→get\_powerControl()**  
**dualpower→powerControl()**  
**dualpower.get\_powerControl()**

**YDualPower**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

function **get\_powerControl( ) As Integer**

**Retourne :**

une valeur parmi Y\_POWERCONTROL\_AUTO, Y\_POWERCONTROL\_FROM\_USB, Y\_POWERCONTROL\_FROM\_EXT et Y\_POWERCONTROL\_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y\_POWERCONTROL\_INVALID.

**dualpower→get\_powerState()**  
**dualpower→powerState()**  
**dualpower.get\_powerState()**

**YDualPower**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

function **get\_powerState( ) As Integer**

**Retourne :**

une valeur parmi Y\_POWERSTATE\_OFF, Y\_POWERSTATE\_FROM\_USB et Y\_POWERSTATE\_FROM\_EXT représentant la source d'alimentation active pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y\_POWERSTATE\_INVALID.

**dualpower→get(userData)**

**YDualPower**

**dualpower→userData()dualpower.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**dualpower→isOnline()****YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le contrôle d'alimentation est joignable, false sinon

**dualpower→load()dualpower.load()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→nextDualPower()**  
**dualpower.nextDualPower()****YDualPower**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

```
function nextDualPower( ) As YDualPower
```

**Retourne :**

un pointeur sur un objet `YDualPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**dualpower→registerValueCallback()  
dualpower.registerValueCallback()****YDualPower**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**dualpower→set\_logicalName()**  
**dualpower→setLogicalName()**  
**dualpower.set\_logicalName()**

**YDualPower**

Modifie le nom logique du contrôle d'alimentation.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**dualpower→set\_powerControl()**  
**dualpower→setPowerControl()**  
**dualpower.set\_powerControl()**

**YDualPower**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

function **set\_powerControl( ByVal newval As Integer) As Integer**

**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).

```
procedure set(userData) ( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.17. Interface de la fonction Files

L'interface de stockage de fichiers permet de stocker des fichiers sur certains modules, par exemple pour personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_files.js'></script>
nodejs var yoctolib = require('yoctolib');
var YFiles = yoctolib.YFiles;
php require_once('yocto_files.php');
cpp #include "yocto_files.h"
m #import "yocto_files.h"
pas uses yocto_files;
vb yocto_files.vb
cs yocto_files.cs
java import com.yoctopuce.YoctoAPI.YFiles;
py from yocto_files import *

```

### Fonction globales

#### **yFindFiles(func)**

Permet de retrouver un système de fichier d'après un identifiant donné.

#### **yFirstFiles()**

Commence l'énumération des système de fichier accessibles par la librairie.

### Méthodes des objets YFiles

#### **files→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **files→download(pathname)**

Télécharge le fichier choisi du filesystème et retourne son contenu.

#### **files→download\_async(pathname, callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

#### **files→format\_fs()**

Rétablissement le système de fichier dans un état original, défragmenté.

#### **files→get\_advertisedValue()**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

#### **files→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### **files→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

#### **files→get\_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

#### **files→get\_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

#### **files→get\_friendlyName()**

Retourne un identifiant global du système de fichier au format NOM\_MODULE . NOM\_FONCTION.

#### **files→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**files→get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

**files→get\_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

**files→get\_list(pattern)**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

**files→get\_logicalName()**

Retourne le nom logique du système de fichier.

**files→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**files→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**files→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**files→isOnline()**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**files→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**files→load(msValidity)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**files→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**files→nextFiles()**

Continue l'énumération des système de fichier commencée à l'aide de yFirstFiles( ).

**files→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**files→remove(pathname)**

Efface un fichier, spécifié par son path complet, du système de fichier.

**files→set\_logicalName(newval)**

Modifie le nom logique du système de fichier.

**files→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**files→upload(pathname, content)**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

**files→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YFiles.FindFiles() yFindFiles()yFindFiles()

YFiles

Permet de retrouver un système de fichier d'après un identifiant donné.

```
function yFindFiles( ByVal func As String) As YFiles
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le système de fichier sans ambiguïté

### Retourne :

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

## YFiles.FirstFiles() yFirstFiles()yFirstFiles()

YFiles

Commence l'énumération des système de fichier accessibles par la librairie.

```
function yFirstFiles( ) As YFiles
```

Utiliser la fonction YFiles.nextFiles( ) pour itérer sur les autres système de fichier.

**Retourne :**

un pointeur sur un objet YFiles, correspondant au premier système de fichier accessible en ligne, ou null si il n'y a pas de système de fichier disponibles.

**files→describe(files.describe())****YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

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()files.download()**

YFiles

Télécharge le fichier choisi du filesystem et retourne son contenu.

```
function download( ) As Byte
```

**Paramètres :**

**pathname** nom complet du fichier à charger, y compris le chemin d'accès.

**Retourne :**

le contenu du fichier chargé sous forme d'objet binaire

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

## files→format\_fs()files.format\_fs()

YFiles

Rétabli le système de fichier dans un état original, défragmenté.

**function format\_fs( ) As Integer**

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).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**files→getErrorMessage()** YFiles  
**files→errorMessage(files.getErrorMessage())**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

**files→get\_errorType()****YFiles****files→errorType()files.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
function get_errorType( ) As YRETCODE
```

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.

<b>files→get_filesCount()</b>	<b>YFiles</b>
<b>files→filesCount()files.get_filesCount()</b>	

---

Retourne le nombre de fichiers présents dans le système de fichier.

```
function get_filesCount( ) As Integer
```

**Retourne :**

un entier représentant le nombre de fichiers présents dans le système de fichier

En cas d'erreur, déclenche une exception ou retourne Y\_FILESCOUNT\_INVALID.

**files→get\_freeSpace()****YFiles****files→freeSpace()files.get\_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

```
function get_freeSpace( ) As Integer
```

**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\_functionDescriptor()**  
**files→functionDescriptor()**  
**files.get\_functionDescriptor()**

**YFiles**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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()****YFiles****files→functionId()files.get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

```
function get_functionId( ) As String
```

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`.

<b>files→get_hardwareId()</b>	<b>YFiles</b>
<b>files→hardwareId(files.get_hardwareId())</b>	

---

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le système de fichier (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**files→get\_list()****YFiles****files→list()files.get\_list()**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
function get_list( ) As List
```

**Paramètres :**

**pattern** un filtre optionnel sur les noms de fichiers retournés, pouvant contenir des astérisques et des points d'interrogations comme jokers. Si le pattern fourni est vide, tous les fichiers sont retournés.

**Retourne :**

une liste d'objets YFileRecord, contenant le nom complet (y compris le chemin d'accès), la taille en octets et le CRC 32-bit du contenu du fichier.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

<b>files→get_logicalName()</b>	<b>YFiles</b>
<b>files→logicalName()files.get_logicalName()</b>	

---

Retourne le nom logique du système de fichier.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du système de fichier.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**files→get\_module()****YFiles****files→module()files.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**files→get(userData)**

**YFiles**

**files→userData(files.get(userData))**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**files→isOnline()files.isOnline()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le système de fichier est joignable, false sinon

**files→load()files.load()****YFiles**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**files→nextFiles()files.nextFiles()****YFiles**

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles()`.

function **nextFiles( )** As YFiles

**Retourne :**

un pointeur sur un objet `YFiles` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**files→registerValueCallback()  
files.registerValueCallback()****YFiles**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**files→remove()files.remove()****YFiles**

Efface un fichier, spécifié par son path complet, du système de fichier.

**function remove( ) As Integer**

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.

---

<b>files→set_logicalName()</b>	<b>YFiles</b>
<b>files→setLogicalName()files.set_logicalName()</b>	

---

Modifie le nom logique du système de fichier.

```
function set_logicalName( ByVal newval As String) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

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.

**procedure upload( )**

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

### Paramètres :

**pathname** nom complet du fichier, y compris le chemin d'accès.

**content** contenu du fichier à télécharger

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.18. Interface de la fonction GenericSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_genericsensor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGenericSensor = yoctolib.YGenericSensor;
php require_once('yocto_genericsensor.php');
cpp #include "yocto_genericsensor.h"
m #import "yocto_genericsensor.h"
pas uses yocto_genericsensor;
vb yocto_genericsensor.vb
cs yocto_genericsensor.cs
java import com.yoctopuce.YoctoAPI.YGenericSensor;
py from yocto_genericsensor import *

```

### Fonction globales

#### **yFindGenericSensor(func)**

Permet de retrouver un capteur générique d'après un identifiant donné.

#### **yFirstGenericSensor()**

Commence l'énumération des capteurs génériques accessibles par la librairie.

### Méthodes des objets **YGenericSensor**

#### **genericsensor→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **genericsensor→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### **genericsensor→get\_advertisedValue()**

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

#### **genericsensor→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

#### **genericsensor→get\_currentValue()**

Retourne la valeur mesurée actuelle.

#### **genericsensor→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### **genericsensor→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

#### **genericsensor→get\_friendlyName()**

Retourne un identifiant global du capteur générique au format NOM\_MODULE . NOM\_FONCTION.

#### **genericsensor→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **genericsensor→get\_functionId()**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

#### **genericsensor→get\_hardwareId()**

### 3. Reference

Retourne l'identifiant matériel unique du capteur générique au format SERIAL . FUNCTIONID.
<b>genericsensor→get_highestValue()</b> Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.
<b>genericsensor→get_logFrequency()</b> Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>genericsensor→get_logicalName()</b> Retourne le nom logique du capteur générique.
<b>genericsensor→get_lowestValue()</b> Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.
<b>genericsensor→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>genericsensor→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>genericsensor→get_recordedData(startTime, endTime)</b> Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>genericsensor→get_reportFrequency()</b> Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>genericsensor→get_resolution()</b> Retourne la résolution des valeurs mesurées.
<b>genericsensor→get_signalBias()</b> Retourne le biais du signal électrique pour la correction du point zéro.
<b>genericsensor→get_signalRange()</b> Retourne la plage de signal électrique utilisée par le capteur.
<b>genericsensor→get_signalUnit()</b> Retourne l'unité du signal électrique utilisée par le capteur.
<b>genericsensor→get_signalValue()</b> Retourne la valeur mesurée du signal électrique utilisée par le capteur.
<b>genericsensor→get_unit()</b> Retourne l'unité dans laquelle la mesure est exprimée.
<b>genericsensor→get(userData)</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>genericsensor→get_valueRange()</b> Retourne la plage de valeurs physiques mesurés par le capteur.
<b>genericsensor→isOnline()</b> Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.
<b>genericsensor→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.
<b>genericsensor→load(msValidity)</b> Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.
<b>genericsensor→loadCalibrationPoints(rawValues, refValues)</b> Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>genericsensor→load_async(msValidity, callback, context)</b>

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

**genericsensor→nextGenericSensor()**

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

**genericsensor→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**genericsensor→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**genericsensor→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

**genericsensor→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**genericsensor→set\_logicalName(newval)**

Modifie le nom logique du capteur générique.

**genericsensor→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**genericsensor→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**genericsensor→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**genericsensor→set\_signalBias(newval)**

Modifie le biais du signal électrique pour la correction du point zéro.

**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.

**genericsensor→zeroAdjust()**

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

## YGenericSensor.FindGenericSensor() yFindGenericSensor()yFindGenericSensor()

**YGenericSensor**

Permet de retrouver un capteur générique d'après un identifiant donné.

```
function yFindGenericSensor( ByVal func As String) As YGenericSensor
```

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()yFirstGenericSensor()

## YGenericSensor

Commence l'énumération des capteurs génériques accessibles par la librairie.

```
function yFirstGenericSensor( ) As YGenericSensor
```

Utiliser la fonction `YGenericSensor.nextGenericSensor()` pour itérer sur les autres capteurs génériques.

### Retourne :

un pointeur sur un objet `YGenericSensor`, correspondant au premier capteur générique accessible en ligne, ou `null` si il n'y a pas de capteurs génériques disponibles.

**genericsensor→calibrateFromPoints()**  
**genericsensor.calibrateFromPoints()****YGenericSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→describe()genericsensor.describe()****YGenericSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format TYPE ( NAME )=SERIAL . FUNCTIONID.

function **describe( )** As String

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur générique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**genericsensor→get\_advertisedValue()**  
**genericsensor→advertisedValue()**  
**genericsensor.get\_advertisedValue()**

---

**YGenericSensor**

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

**function get\_advertisedValue( ) As String**

**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\_ADVISEDVALUE\_INVALID.

**genericsensor→get\_currentRawValue()**  
**genericsensor→currentRawValue()**  
**genericsensor.get\_currentRawValue()**

**YGenericSensor**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**genericsensor→get\_currentValue()**  
**genericsensor→currentValue()**  
**genericsensor.get\_currentValue()**

---

**YGenericSensor**

Retourne la valeur mesurée actuelle.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**genericsensor→getErrorMessage()**  
**genericsensor→errorMessage()**  
**genericsensor.getErrorMessage()**

**YGenericSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

**function getErrorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

**genericsensor→get\_errorType()**  
**genericsensor→errorType()**  
**genericsensor.get\_errorType()**

**YGenericSensor**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

**function get\_errorType( ) As YRETCODE**

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\_functionDescriptor()**  
**genericsensor→functionDescriptor()**  
**genericsensor.get\_functionDescriptor()**

**YGenericSensor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**genericsensor→get\_functionId()**  
**genericsensor→functionId()**  
**genericsensor.get\_functionId()**

---

**YGenericSensor**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

function **get\_functionId( ) As String**

Par example `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**genericsensor→get\_hardwareId()**  
**genericsensor→hardwareId()**  
**genericsensor.get\_hardwareId()**

**YGenericSensor**

Retourne l'identifiant matériel unique du capteur générique au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur générique (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur générique (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**genericsensor→get\_highestValue()**  
**genericsensor→highestValue()**  
**genericsensor.get\_highestValue()**

**YGenericSensor**

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

**genericsensor→get\_logFrequency()**  
**genericsensor→logFrequency()**  
**genericsensor.get\_logFrequency()**

**YGenericSensor**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

function **get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne **Y\_LOGFREQUENCY\_INVALID**.

**genericsensor→get\_logicalName()**  
**genericsensor→logicalName()**  
**genericsensor.get\_logicalName()**

---

**YGenericSensor**

Retourne le nom logique du capteur générique.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur générique.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**genericsensor→get\_lowestValue()**  
**genericsensor→lowestValue()**  
**genericsensor.get\_lowestValue()**

**YGenericSensor**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

```
function get_lowestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**genericsensor→get\_module()**  
**genericsensor→module()**  
**genericsensor.get\_module()**

---

**YGenericSensor**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**genericsensor→get\_recordedData()**  
**genericsensor→recordedData()**  
**genericsensor.get\_recordedData()**

**YGenericSensor**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**genericsensor→get\_reportFrequency()**  
**genericsensor→reportFrequency()**  
**genericsensor.get\_reportFrequency()**

**YGenericSensor**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**genericsensor→get\_resolution()**  
**genericsensor→resolution()**  
**genericsensor.get\_resolution()**

**YGenericSensor**

Retourne la résolution des valeurs mesurées.

**function get\_resolution( ) As Double**

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\_signalBias()**  
**genericsensor→signalBias()**  
**genericsensor.get\_signalBias()**

---

**YGenericSensor**

Retourne le biais du signal électrique pour la correction du point zéro.

**function get\_signalBias( ) As Double**

Un biais positif correspond à la correction d'un signal trop positif, tandis qu'un biais négatif correspond à la correction d'un signal trop négatif.

**Retourne :**

une valeur numérique représentant le biais du signal électrique pour la correction du point zéro

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALBIAS\_INVALID.

**genericsensor→get\_signalRange()**  
**genericsensor→signalRange()**  
**genericsensor.get\_signalRange()**

**YGenericSensor**

Retourne la plage de signal électrique utilisée par le capteur.

```
function get_signalRange( ) As String
```

**Retourne :**

une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALRANGE\_INVALID.

**genericsensor→get\_signalUnit()**  
**genericsensor→signalUnit()**  
**genericsensor.get\_signalUnit()**

---

**YGenericSensor**

Retourne l'unité du signal électrique utilisée par le capteur.

```
function get_signalUnit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALUNIT\_INVALID.

**genericsensor→get\_signalValue()**  
**genericsensor→signalValue()**  
**genericsensor.get\_signalValue()**

**YGenericSensor**

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

```
function get_signalValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur mesurée du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y\_SIGNALVALUE\_INVALID.

**genericsensor→get\_unit()**

**YGenericSensor**

**genericsensor→unit()genericsensor.get\_unit()**

---

Retourne l'unité dans laquelle la mesure est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**genericsensor→get(userData)**  
**genericsensor→userData()**  
**genericsensor.get(userData)**

**YGenericSensor**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**genericsensor→get\_valueRange()**  
**genericsensor→valueRange()**  
**genericsensor.get\_valueRange()**

**YGenericSensor**

---

Retourne la plage de valeurs physiques mesurés par le capteur.

function **get\_valueRange( ) As String**

**Retourne :**

une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

En cas d'erreur, déclenche une exception ou retourne **Y\_VALUERANGE\_INVALID**.

**genericsensor→isOnline()genericsensor.isOnline()****YGenericSensor**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du capteur générique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur générique est joignable, false sinon

**genericsensor→load()genericsensor.load()****YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

```
procedure loadCalibrationPoints( )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→nextGenericSensor()**  
**genericsensor.nextGenericSensor()**

---

**YGenericSensor**

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

```
function nextGenericSensor( ) As YGenericSensor
```

**Retourne :**

un pointeur sur un objet `YGenericSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**genericsensor→registerTimedReportCallback()  
genericsensor.registerTimedReportCallback()****YGenericSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**genericsensor→set\_highestValue()**  
**genericsensor→setHighestValue()**  
**genericsensor.set\_highestValue()**

**YGenericSensor**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_logFrequency()**  
**genericsensor→setLogFrequency()**  
**genericsensor.set\_logFrequency()**

**YGenericSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_logicalName()**  
**genericsensor→setLogicalName()**  
**genericsensor.set\_logicalName()**

**YGenericSensor**

Modifie le nom logique du capteur générique.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du capteur générique.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_lowestValue()**  
**genericsensor→setLowestValue()**  
**genericsensor.set\_lowestValue()**

**YGenericSensor**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_reportFrequency()**  
**genericsensor→setReportFrequency()**  
**genericsensor.set\_reportFrequency()**

**YGenericSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_resolution()**  
**genericsensor→setResolution()**  
**genericsensor.set\_resolution()**

**YGenericSensor**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( ByVal newval As Double) As Integer**

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\_signalBias()**  
**genericsensor→setSignalBias()**  
**genericsensor.set\_signalBias()**

**YGenericSensor**

Modifie le biais du signal électrique pour la correction du point zéro.

```
function set_signalBias( ByVal newval As Double) As Integer
```

Si votre signal électrique est positif lorsqu'il devrait être nul, configurez un biais positif de la même valeur afin de corriger l'erreur.

**Paramètres :**

**newval** une valeur numérique représentant le biais du signal électrique pour la correction du point zéro

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_signalRange()**  
**genericsensor→setSignalRange()**  
**genericsensor.set\_signalRange()**

**YGenericSensor**

Modifie la plage de signal électrique utilisée par le capteur.

function **set\_signalRange( ByVal newval As String) As Integer**

**Paramètres :**

**newval** une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**genericsensor→set\_unit()****YGenericSensor****genericsensor→setUnit()genericsensor.set\_unit()**

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
function set_unit( ByVal newval As String) As Integer
```

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).

**procedure set(userData( ByVal data As Object)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**genericsensor→set\_valueRange()**  
**genericsensor→setValueRange()**  
**genericsensor.set\_valueRange()**

**YGenericSensor**

Modifie la plage de valeurs physiques mesurés par le capteur.

```
function set_valueRange( ByVal newval As String) As Integer
```

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→zeroAdjust()**  
**genericsensor.zeroAdjust()**

---

**YGenericSensor**

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

function **zeroAdjust( ) As Integer**

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

## 3.19. Interface de la fonction Gyro

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_gyro.js'></script>
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

#### 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 renvoyée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule.

#### gyro→get\_currentValue()

Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.

#### 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()

### 3. Reference

Retourne l'identifiant matériel unique du gyroscope au format SERIAL.FUNCTIONID.
<b>gyro→get_heading()</b> Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.
<b>gyro→get_highestValue()</b> Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.
<b>gyro→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>gyro→get_logicalName()</b> Retourne le nom logique du gyroscope.
<b>gyro→get_lowestValue()</b> Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.
<b>gyro→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>gyro→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>gyro→get_pitch()</b> Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.
<b>gyro→get_quaternionW()</b> 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.
<b>gyro→get_quaternionX()</b> 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.
<b>gyro→get_quaternionY()</b> 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.
<b>gyro→get_quaternionZ()</b> Retourne la composante z du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.
<b>gyro→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>gyro→get_reportFrequency()</b> Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>gyro→get_resolution()</b> Retourne la résolution des valeurs mesurées.
<b>gyro→get_roll()</b> Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.
<b>gyro→get_unit()</b> Retourne l'unité dans laquelle la vitesse angulaire est exprimée.
<b>gyro→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**gyro→get\_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

**gyro→get\_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

**gyro→get\_zValue()**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

**gyro→isOnline()**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

**gyro→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

**gyro→load(msValidity)**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

**gyro→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

**gyro→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

**gyro→nextGyro()**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

**gyro→registerAnglesCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

**gyro→registerQuaternionCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

**gyro→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**gyro→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**gyro→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

**gyro→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**gyro→set\_logicalName(newval)**

Modifie le nom logique du gyroscope.

**gyro→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**gyro→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**gyro→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**gyro→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**gyro→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YGyro.FindGyro() yFindGyro()yFindGyro()

YGyro

Permet de retrouver un gyroscope d'après un identifiant donné.

```
function yFindGyro( ByVal func As String) As YGyro
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le gyroscope sans ambiguïté

### Retourne :

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

## YGyro.FirstGyro() yFirstGyro()yFirstGyro()

YGyro

Commence l'énumération des gyroscopes accessibles par la librairie.

```
function yFirstGyro( ) As YGyro
```

Utiliser la fonction `YGyro.nextGyro()` pour itérer sur les autres gyroscopes.

**Retourne :**

un pointeur sur un objet `YGyro`, correspondant au premier gyroscope accessible en ligne, ou `null` si il n'y a pas de gyroscopes disponibles.

**gyro→calibrateFromPoints()  
gyro.calibrateFromPoints()****YGyro**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→describe()gyro.describe()****YGyro**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le gyroscope (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

<b>gyro→get_advertisedValue()</b>	<b>YGyro</b>
<b>gyro→advertisedValue()gyro.get_advertisedValue()</b>	

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du gyroscope (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**gyro→get\_currentRawValue()****YGyro****gyro→currentRawValue()gyro.get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule.

```
function get_currentRawValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

<b>gyro→get_currentValue()</b>	<b>YGyro</b>
<b>gyro→currentValue()gyro.get_currentValue()</b>	

Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**gyro→get\_errorMessage()****YGyro****gyro→errorMessage()gyro.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
function get_errorMessage( ) As String
```

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.

<b>gyro→get_errorType()</b>	<b>YGyro</b>
<b>gyro→errorType()gyro.get_errorType()</b>	

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
function get_errorType( ) As YRETCODE
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

**gyro→get\_functionDescriptor()**  
**gyro→functionDescriptor()**  
**gyro.get\_functionDescriptor()****YGyro**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

<b>gyro→get_functionId()</b>	<b>YGyro</b>
<b>gyro→functionId()gyro.get_functionId()</b>	

---

Retourne l'identifiant matériel du gyroscope, sans référence au module.

```
function get_functionId( ) As String
```

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.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le gyroscope (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

<b>gyro→get_heading()</b>	<b>YGyro</b>
<b>gyro→heading()gyro.get_heading()</b>	

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**function get\_heading( ) As Double**

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.

function **get\_highestValue( ) As Double**

**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.

<b>gyro→get_logFrequency()</b>	<b>YGyro</b>
<b>gyro→logFrequency()gyro.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.

```
function get_logFrequency( ) As String
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**gyro→get\_logicalName()****YGyro****gyro→logicalName()gyro.get\_logicalName()**

Retourne le nom logique du gyroscope.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du gyroscope.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**gyro→get\_lowestValue()**

**YGyro**

**gyro→lowestValue()gyro.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**gyro→get\_module()****YGyro****gyro→module()gyro.get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule rentrée ne sera pas joignable.

**Retourne :**

une instance de YModule

<b>gyro→get_pitch()</b>	<b>YGyro</b>
<b>gyro→pitch()gyro.get_pitch()</b>	

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**function get\_pitch( ) As Double**

L'axe de tangage peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe YRefFrame.

**Retourne :**

un nombre à virgule correspondant à l'assiette, exprimée en degrés (entre -90 et +90).

**gyro→get\_quaternionW()****YGyro****gyro→quaternionW()gyro.get\_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_quaternionW( ) As Double
```

**Retourne :**

un nombre à virgule correspondant à la composante w du quaternion.

---

<b>gyro→get_quaternionX()</b>	<b>YGyro</b>
<b>gyro→quaternionX()gyro.get_quaternionX()</b>	

---

Retourne la composante `x` du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_quaternionX( ) As Double
```

La composante `x` est essentiellement corrélée aux rotations sur l'axe de roulis.

**Retourne :**

un nombre à virgule correspondant à la composante `x` du quaternion.

---

**gyro→get\_quaternionY()****YGyro****gyro→quaternionY()gyro.get\_quaternionY()**

Retourne la composante  $y$  du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_quaternionY( ) As Double
```

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.

**function get\_quaternionZ( ) As Double**

La composante  $z$  est essentiellement corrélée aux rotations sur l'axe de lacet.

**Retourne :**

un nombre à virgule correspondant à la composante  $z$  du quaternion.

**gyro→get\_recordedData()****YGyro****gyro→recordedData()gyro.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( ) As YDataSet
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

---

<b>gyro→get_reportFrequency()</b>	<b>YGyro</b>
<b>gyro→reportFrequency()gyro.get_reportFrequency()</b>	

---

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( ) As String
```

**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.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

<b>gyro→get_roll()</b>	<b>YGyro</b>
<b>gyro→roll()gyro.get_roll()</b>	

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

**function get\_roll( ) As Double**

L'axe de roulis peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe YRefFrame.

**Retourne :**

un nombre à virgule correspondant à l'inclinaison, exprimée en degrés (entre -180 et +180).

**gyro→get\_unit()****YGyro****gyro→unit()gyro.get\_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

```
function get_unit( ) As String
```

**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.

<b>gyro→get(userData)</b>	<b>YGyro</b>
<b>gyro→userData()gyro.get(userData)</b>	

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**gyro→get\_xValue()****YGyro****gyro→xValue()gyro.get\_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

```
function get_xValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_XVALUE\_INVALID.

**gyro→get\_yValue()**

**YGyro**

**gyro→yValue()gyro.get\_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

**function get\_yValue( ) As Double**

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_YVALUE\_INVALID.

**gyro→get\_zValue()****YGyro****gyro→zValue()gyro.get\_zValue()**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

```
function get_zValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_ZVALUE\_INVALID.

**gyro→isOnline()gyro.isOnline()****YGyro**

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le gyroscope est joignable, false sinon

## gyro→load()gyro.load()

## YGyro

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

procedure **loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→nextGyro()gyro.nextGyro()****YGyro**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

```
function nextGyro() As YGyro
```

**Retourne :**

un pointeur sur un objet `YGyro` accessible en ligne, ou `null` lorsque l'énumération est terminée.

---

<b>gyro→registerAnglesCallback()</b>	<b>YGyro</b>
<b>gyro.registerAnglesCallback()</b>	

---

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

**function registerAnglesCallback( ) As Integer**

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appellé trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter quatre arguments: l'objet YGyro du module qui a tourné, et les valeurs des trois angles roll, pitch et heading en degrés (nombres à virgules).

**gyro→registerQuaternionCallback()**  
**gyro.registerQuaternionCallback()****YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
function registerQuaternionCallback( ) As Integer
```

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.

```
function registerTimedReportCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**gyro→registerValueCallback()**  
**gyro.registerValueCallback()****YGyro**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

<b>gyro→set_highestValue()</b>	<b>YGyro</b>
<b>gyro→setHighestValue()gyro.set_highestValue()</b>	

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_logFrequency()****YGyro****gyro→setLogFrequency()gyro.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>gyro→set_logicalName()</b>	<b>YGyro</b>
<b>gyro→setLogicalName()gyro.set_logicalName()</b>	

---

Modifie le nom logique du gyroscope.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du gyroscope.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**gyro→set\_lowestValue()****YGyro****gyro→setLowestValue()gyro.set\_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_reportFrequency()**  
**gyro→setReportFrequency()**  
**gyro.set\_reportFrequency()**

**YGyro**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**gyro→set\_resolution()****YGyro****gyro→setResolution()gyro.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>gyro→set(userData)</b>	<b>YGyro</b>
<b>gyro→setUserData()gyro.set(userData)</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).

```
procedure set(userData) ByVal data As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.20. Interface 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()yFindHubPort()

YHubPort

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

```
function yFindHubPort( ByVal func As String) As YHubPort
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port de Yocto-hub soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHubPort.isOnline()` pour tester si le port de Yocto-hub est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le port de Yocto-hub sans ambiguïté

### Retourne :

un objet de classe `YHubPort` qui permet ensuite de contrôler le port de Yocto-hub.

## **YHubPort.FirstHubPort() yFirstHubPort()yFirstHubPort()**

**YHubPort**

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

```
function yFirstHubPort( ) As YHubPort
```

Utiliser la fonction `YHubPort.nextHubPort( )` pour itérer sur les autres port de Yocto-hub.

**Retourne :**

un pointeur sur un objet `YHubPort`, correspondant au premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

**hubport→describe()hubport.describe()****YHubPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **describe( )** As String

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un débuggeur.

**Retourne :**

une chaîne de caractères décrivant le port de Yocto-hub (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**hubport→get\_advertisedValue()**  
**hubport→advertisedValue()**  
**hubport.get\_advertisedValue()**

**YHubPort**

---

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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_ADVISEDVALUE_INVALID`.

**hubport→get\_baudRate()****YHubPort****hubport→baudRate()hubport.get\_baudRate()**

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

```
function get_baudRate( ) As Integer
```

La valeur par défaut est 1000 kbps, une valeur inférieure révèle des problèmes de communication.

**Retourne :**

un entier représentant la vitesse de transfert utilisée par le port de Yocto-hub, en kbps

En cas d'erreur, déclenche une exception ou retourne Y\_BAUDRATE\_INVALID.

**hubport→get\_enabled()**

**YHubPort**

**hubport→enabled()hubport.get\_enabled()**

---

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

```
function get_enabled( ) As Integer
```

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon vrai si le port du Yocto-hub est alimenté,  
faux sinon

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**hubport→getErrorMessage()****YHubPort****hubport→errorMessage()hubport.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

**hubport→get\_errorType()**

**YHubPort**

**hubport→errorType()hubport.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**hubport→functionDescriptor()**  
**hubport.get\_functionDescriptor()**

**YHubPort**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

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.

function **get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**hubport→get\_hardwareId()****YHubPort****hubport→hardwareId()hubport.get\_hardwareId()**

Retourne l'identifiant matériel unique du port de Yocto-hub au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port de Yocto-hub (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le port de Yocto-hub (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**hubport→get\_logicalName()**

**YHubPort**

**hubport→logicalName()hubport.get\_logicalName()**

---

Retourne le nom logique du port de Yocto-hub.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port de Yocto-hub.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**hubport→get\_module()****YHubPort****hubport→module()hubport.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**hubport→get\_portState()** YHubPort  
**hubport→portState()hubport.get\_portState()**

---

Retourne l'état actuel du port de Yocto-hub.

```
function get_portState( ) As Integer
```

**Retourne :**

une valeur parmi Y\_PORTSTATE\_OFF, Y\_PORTSTATE\_OVRLD, Y\_PORTSTATE\_ON,  
Y\_PORTSTATE\_RUN et Y\_PORTSTATE\_PROG représentant l'état actuel du port de Yocto-hub

En cas d'erreur, déclenche une exception ou retourne Y\_PORTSTATE\_INVALID.

**hubport→get(userData)****YHubPort****hubport→userData()hubport.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

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.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du port de Yocto-hub sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le port de Yocto-hub est joignable, false sinon

**hubport→load()hubport.load()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **hubport→nextHubPort()hubport.nextHubPort()**

**YHubPort**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

```
function nextHubPort( ) As YHubPort
```

**Retourne :**

un pointeur sur un objet `YHubPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**hubport→registerValueCallback()**  
**hubport.registerValueCallback()****YHubPort**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

<b>hubport→set_enabled()</b>	<b>YHubPort</b>
<b>hubport→setEnabled()hubport.set_enabled()</b>	

---

Modifie le mode d'activation du port du Yocto-hub.

```
function set_enabled( ByVal newval As Integer) As Integer
```

Si le port est actif, il sera alimenté. Sinon, l'alimentation du module est coupée.

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon le mode d'activation du port du Yocto-hub

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**hubport→set\_logicalName()**  
**hubport→setLogicalName()**  
**hubport.set\_logicalName()**

**YHubPort**

Modifie le nom logique du port de Yocto-hub.

```
function set_logicalName( ByVal newval As String) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.21. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_humidity.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YHumidity = yoctolib.YHumidity;
php	require_once('yocto_humidity.php');
cpp	#include "yocto_humidity.h"
m	#import "yocto_humidity.h"
pas	uses yocto_humidity;
vb	yocto_humidity.vb
cs	yocto_humidity.cs
java	import com.yoctopuce.YoctoAPI.YHumidity;
py	from yocto_humidity import *

### Fonction globales

#### yFindHumidity(func)

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

#### yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

### Méthodes des objets YHumidity

#### humidity→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### humidity→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### humidity→get\_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

#### humidity→get\_currentRawValue()

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule.

#### humidity→get\_currentValue()

Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.

#### humidity→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### humidity→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

#### humidity→get\_friendlyName()

Retourne un identifiant global du capteur d'humidité au format NOM\_MODULE . NOM\_FONCTION.

#### humidity→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### humidity→get\_functionId()

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

#### humidity→get\_hardwareId()

### 3. Reference

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.
<b>humidity→get_highestValue()</b> Retourne la valeur maximale observée pour l'humidité depuis le démarrage du module.
<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 depuis le démarrage du module.
<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.
<b>humidity→set_logFrequency(newval)</b>

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**humidity→set\_logicalName(newval)**

Modifie le nom logique du capteur d'humidité.

**humidity→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**humidity→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**humidity→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**humidity→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**humidity→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YHumidity.FindHumidity() yFindHumidity()yFindHumidity()

YHumidity

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

```
function yFindHumidity( ByVal func As String) As YHumidity
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur d'humidité soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnLine()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

### Retourne :

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

**YHumidity.FirstHumidity()****yFirstHumidity()yFirstHumidity()****YHumidity**

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

```
function yFirstHumidity( ) As YHumidity
```

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

**Retourne :**

un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou null si il n'y a pas de capteurs d'humidité disponibles.

**humidity→calibrateFromPoints()****YHumidity****humidity.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→describe()humidity.describe()****YHumidity**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **describe( )** As String

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur d'humidité (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**humidity→get\_advertisedValue()**

**YHumidity**

**humidity→advertisedValue()**

**humidity.get\_advertisedValue()**

---

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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\_ADVISEDVALUE\_INVALID.

**humidity→get\_currentRawValue()**  
**humidity→currentRawValue()**  
**humidity.get\_currentRawValue()**

**YHumidity**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

**humidity→get\_currentValue()**

**YHumidity**

**humidity→currentValue()humidity.get\_currentValue()**

---

Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**humidity→getErrorMessage()**  
**humidity→errorMessage()**  
**humidity.getErrorMessage()****YHumidity**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

## humidity→get\_errorType()

YHumidity

## humidity→errorType()humidity.get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**humidity→functionDescriptor()**  
**humidity.get\_functionDescriptor()****YHumidity**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**humidity→get\_functionId()**

**YHumidity**

**humidity→functionId()humidity.get\_functionId()**

---

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur d'humidité (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**humidity→get\_hardwareId()****YHumidity****humidity→hardwareId()humidity.get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

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 depuis le démarrage du module.

**function get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'humidité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**humidity→get\_logFrequency()**  
**humidity→logFrequency()**  
**humidity.get\_logFrequency()**

**YHumidity**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

function **get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne **Y\_LOGFREQUENCY\_INVALID**.

**humidity→get\_logicalName()**

**YHumidity**

**humidity→logicalName()humidity.get\_logicalName()**

---

Retourne le nom logique du capteur d'humidité.

```
function get_logicalName( ) As String
```

**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 depuis le démarrage du module.

```
function get_lowestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'humidité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

## humidity→get\_module()

YHumidity

## humidity→module()humidity.get\_module()

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**humidity→get\_recordedData()**  
**humidity→recordedData()**  
**humidity.get\_recordedData()****YHumidity**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( ) As YDataSet
```

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**humidity→get\_reportFrequency()**  
**humidity→reportFrequency()**  
**humidity.get\_reportFrequency()**

**YHumidity**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne **Y\_REPORTFREQUENCY\_INVALID**.

**humidity→get\_resolution()****YHumidity****humidity→resolution()humidity.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**humidity→get\_unit()**

**YHumidity**

**humidity→unit()humidity.get\_unit()**

---

Retourne l'unité dans laquelle l'humidité est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'humidité est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**humidity→get(userData)****YHumidity****humidity→userData()humidity.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**humidity→isOnline()**humidity.isOnline()******YHumidity**

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du capteur d'humidité sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur d'humidité est joignable, `false` sinon

**humidity→load()**humidity.load()******YHumidity**

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

procedure **loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→nextHumidity()humidity.nextHumidity()****YHumidity**

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

```
function nextHumidity( ) As YHumidity
```

**Retourne :**

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**humidity→registerTimedReportCallback()  
humidity.registerTimedReportCallback()****YHumidity**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**humidity→registerValueCallback()**  
**humidity.registerValueCallback()****YHumidity**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

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.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_logFrequency()  
humidity→setLogFrequency()  
humidity.set\_logFrequency()****YHumidity**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_logicalName()**  
**humidity→setLogicalName()**  
**humidity.set\_logicalName()**

**YHumidity**

Modifie le nom logique du capteur d'humidité.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur d'humidité.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_lowestValue()**  
**humidity→setLowestValue()**  
**humidity.set\_lowestValue()**

**YHumidity**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>humidity→set_reportFrequency()</b>	<b>YHumidity</b>
<b>humidity→setReportFrequency()</b>	
<b>humidity.set_reportFrequency()</b>	

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**humidity→set\_resolution()****YHumidity****humidity→setResolution()humidity.set\_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

```
procedure set(userData)( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.22. Interface de la fonction 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()yFindLed()

YLed

Permet de retrouver une led d'après un identifiant donné.

```
function yFindLed( ByVal func As String) As YLed
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la led soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLed.isOnline()` pour tester si la led est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence la led sans ambiguïté

**Retourne :**

un objet de classe `YLed` qui permet ensuite de contrôler la led.

## YLed.FirstLed() yFirstLed()yFirstLed()

YLed

Commence l'énumération des leds accessibles par la librairie.

```
function yFirstLed( ) As YLed
```

Utiliser la fonction `YLed.nextLed()` pour itérer sur les autres leds.

**Retourne :**

un pointeur sur un objet `YLed`, correspondant à la première led accessible en ligne, ou `null` si il n'y a pas de leds disponibles.

**led→describe()led.describe()****YLed**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format TYPE ( NAME )=SERIAL.FUNCTIONID.

function **describe( )** As String

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant la led (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**led→get\_advertisedValue()**

YLed

**led→advertisedValue()led.get\_advertisedValue()**

---

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**led→get\_blinking()****YLed****led→blinking()led.get\_blinking()**

Retourne le mode de signalisation de la led.

```
function get_blinking( ) As Integer
```

**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.

**function get\_errorMessage( ) As String**

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.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**led→functionDescriptor()**  
**led.get\_functionDescriptor()**

YLed

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**led→get\_functionId()****YLed****led→functionId()led.get\_functionId()**

Retourne l'identifiant matériel de la led, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant la led (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**led→get\_hardwareId()**

YLed

**led→hardwareId()led.get\_hardwareId()**

---

Retourne l'identifiant matériel unique de la led au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la led (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**led→get\_logicalName()****YLed****led→logicalName()led.get\_logicalName()**

Retourne le nom logique de la led.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de la led.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**led→get\_luminosity()**

YLed

**led→luminosity()led.get\_luminosity()**

---

Retourne l'intensité de la led en pour cent.

```
function get_luminosity( ) As Integer
```

**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.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**led→get\_power()**

YLed

**led→power()led.get\_power()**

---

Retourne l'état courant de la led.

```
function get_power( ) As Integer
```

**Retourne :**

soit Y\_POWER\_OFF, soit Y\_POWER\_ON, selon l'état courant de la led

En cas d'erreur, déclenche une exception ou retourne Y\_POWER\_INVALID.

**led→get(userData)****YLed****led→userData()led.get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**led→isOnline()led.isOnline()**

YLed

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache de la led sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si la led est joignable, false sinon

**led→load()led.load()****YLed**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## led→nextLed()led.nextLed()

YLed

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

```
function nextLed( ) As YLed
```

**Retourne :**

un pointeur sur un objet YLed accessible en ligne, ou `null` lorsque l'énumération est terminée.

**led→registerValueCallback()  
led.registerValueCallback()****YLed**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**led->set\_blinking()** YLed  
**led->setBlinking()led.set\_blinking()**

Modifie le mode de signalisation de la led.

```
function set_blinking( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** une valeur parmi Y\_BLINKING\_STILL, Y\_BLINKING\_RELAX, Y\_BLINKING\_AWARE, Y\_BLINKING\_RUN, Y\_BLINKING\_CALL et Y\_BLINKING\_PANIC représentant le mode de signalisation de la led

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→set\_logicalName()****YLed****led→setLogicalName()led.set\_logicalName()**

Modifie le nom logique de la led.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la led.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## led->set\_luminosity()

YLed

## led->setLuminosity()led.set\_luminosity()

---

Modifie l'intensité lumineuse de la led (en pour cent).

```
function set_luminosity( ByVal newval As Integer) As Integer
```

### Paramètres :

**newval** un entier représentant l'intensité lumineuse de la led (en pour cent)

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**led→set\_power()****YLed****led→setPower()|led.set\_power()**

Modifie l'état courant de la led.

```
function set_power( ByVal newval As Integer) As Integer
```

**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)**  
**led→setUserData()|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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.23. Interface de la fonction LightSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_lightsensor.js'></script>
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 renvoyée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule.

#### lightsensor→get\_currentValue()

Retourne la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule.

#### 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.

<b>lightsensor→get_functionId()</b>	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 depuis le démarrage du module.
<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 depuis le démarrage du module.
<b>lightsensor→get_measureType()</b>	Retourne le type de mesure de lumière utilisé par le module.
<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.

**lightsensor→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**lightsensor→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

**lightsensor→set\_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.

**lightsensor→set\_measureType(newval)**

Change le type dde mesure de lumière effectuée par le capteur.

**lightsensor→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**lightsensor→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**lightsensor→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**lightsensor→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YLightSensor.FindLightSensor() yFindLightSensor()yFindLightSensor()

YLightSensor

Permet de retrouver un capteur de lumière d'après un identifiant donné.

```
function yFindLightSensor( ByVal func As String) As YLightSensor
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de lumière soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLightSensor.isOnline()` pour tester si le capteur de lumière est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le capteur de lumière sans ambiguïté

### Retourne :

un objet de classe `YLightSensor` qui permet ensuite de contrôler le capteur de lumière.

**YLightSensor.FirstLightSensor()****yFirstLightSensor()yFirstLightSensor()****YLightSensor**

Commence l'énumération des capteurs de lumière accessibles par la librairie.

```
function yFirstLightSensor( ) As YLightSensor
```

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

**Retourne :**

un pointeur sur un objet `YLightSensor`, correspondant au premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

**lightsensor→calibrate()|lightsensor.calibrate()****YLightSensor**

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

```
function calibrate( ByVal calibratedVal As Double) As Integer
```

**Paramètres :**

**calibratedVal** la consigne de valeur désirée.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→calibrateFromPoints()**  
**lightsensor.calibrateFromPoints()****YLightSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

procedure **calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→describe()lightsensor.describe()****YLightSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe() As String
```

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).

```
function get_advertisedValue( ) As String
```

**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), en lux, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

**lightsensor→get\_currentValue()**  
**lightsensor→currentValue()**  
**lightsensor.get\_currentValue()**

**YLightSensor**

Retourne la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**lightsensor→getErrorMessage()**  
**lightsensor→errorMessage()**  
**lightsensor.getErrorMessage()**

**YLightSensor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

**function getErrorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

**lightsensor→get\_errorType()****YLightSensor****lightsensor→errorType()lightsensor.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**lightsensor→functionDescriptor()**  
**lightsensor.get\_functionDescriptor()**

---

**YLightSensor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**lightsensor→get\_functionId()****YLightSensor****lightsensor→functionId()lightsensor.get\_functionId()**

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

```
function get_functionId( ) As String
```

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.

**function get\_hardwareId( ) As String**

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 depuis le démarrage du module.

```
function get_highestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la lumière ambiante depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**lightsensor→get\_logFrequency()**  
**lightsensor→logFrequency()**  
**lightsensor.get\_logFrequency()**

**YLightSensor**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

function **get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**lightsensor→get\_logicalName()**  
**lightsensor→logicalName()**  
**lightsensor.get\_logicalName()****YLightSensor**

Retourne le nom logique du capteur de lumière.

```
function get_logicalName( ) As String
```

**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 depuis le démarrage du module.

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**lightsensor→get\_measureType()**  
**lightsensor→measureType()**  
**lightsensor.get\_measureType()**

**YLightSensor**

Retourne le type de mesure de lumière utilisé par le module.

```
function get_measureType( ) As Integer
```

**Retourne :**

une valeur parmi Y\_MEASURETYPE\_HUMAN\_EYE, Y\_MEASURETYPE\_WIDE\_SPECTRUM, Y\_MEASURETYPE\_INFRARED, Y\_MEASURETYPE\_HIGH\_RATE et Y\_MEASURETYPE\_HIGH\_ENERGY représentant le type de mesure de lumière utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_MEASURETYPE\_INVALID.

**lightsensor→get\_module()**

**YLightSensor**

**lightsensor→module()lightsensor.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**lightsensor→get\_recordedData()**  
**lightsensor→recordedData()**  
**lightsensor.get\_recordedData()****YLightSensor**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( ) As YDataSet
```

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**lightsensor→get\_reportFrequency()**  
**lightsensor→reportFrequency()**  
**lightsensor.get\_reportFrequency()**

**YLightSensor**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**lightsensor→get\_resolution()****YLightSensor****lightsensor→resolution()lightsensor.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**lightsensor→get\_unit()**

**YLightSensor**

**lightsensor→unit()lightsensor.get\_unit()**

---

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la lumière ambiante est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**lightsensor→get(userData)****YLightSensor****lightsensor→userData()lightsensor.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## **lightsensor→isOnline()lightsensor.isOnline()**

**YLightSensor**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du capteur de lumière sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de lumière est joignable, false sinon

**lightsensor→load()lightsensor.load()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→loadCalibrationPoints()**  
**lightsensor.loadCalibrationPoints()****YLightSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

procedure **loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→nextLightSensor()****YLightSensor****lightsensor.nextLightSensor()**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

```
function nextLightSensor( ) As YLightSensor
```

**Retourne :**

un pointeur sur un objet `YLightSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**lightsensor→registerTimedReportCallback()**  
**lightsensor.registerTimedReportCallback()****YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

**lightsensor→registerValueCallback()  
lightsensor.registerValueCallback()****YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**lightsensor→set\_highestValue()**  
**lightsensor→setHighestValue()**  
**lightsensor.set\_highestValue()**

**YLightSensor**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logFrequency()**  
**lightsensor→setLogFrequency()**  
**lightsensor.set\_logFrequency()****YLightSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_logicalName()**  
**lightsensor→setLogicalName()**  
**lightsensor.set\_logicalName()**

**YLightSensor**

Modifie le nom logique du capteur de lumière.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_measureType()**  
**lightsensor→setMeasureType()**  
**lightsensor.set\_measureType()**

**YLightSensor**

Change le type dde mesure de lumière effectuée par le capteur.

```
function set_measureType( ByVal newval As Integer) As Integer
```

La mesure peut soit approximer la réponse de l'oeil humain, soit donner une valeur ciblant un spectre particulier, en fonction des possibilités offertes par le récepteur de lumière. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur parmi Y\_MEASURETYPE\_HUMAN\_EYE,  
Y\_MEASURETYPE\_WIDE\_SPECTRUM, Y\_MEASURETYPE\_INFRARED,  
Y\_MEASURETYPE\_HIGH\_RATE et Y\_MEASURETYPE\_HIGH\_ENERGY

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_reportFrequency()**  
**lightsensor→setReportFrequency()**  
**lightsensor.set\_reportFrequency()****YLightSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**lightsensor→set\_resolution()**  
**lightsensor→setResolution()**  
**lightsensor.set\_resolution()**

**YLightSensor**

Modifie la résolution des valeurs physique mesurées.

**function set\_resolution( ByVal newval As Double) As Integer**

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).

procedure **set(userData)** ByVal **data** As Object

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.24. 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;
require_once('yocto_magnetometer.php');
#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 rentrée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule.

#### **magnetometer→get\_currentValue()**

Retourne la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule.

#### **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>

### 3. Reference

---

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**magnetometer→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**magnetometer→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

**magnetometer→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**magnetometer→set\_logicalName(newval)**

Modifie le nom logique du magnétomètre.

**magnetometer→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**magnetometer→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**magnetometer→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**magnetometer→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**magnetometer→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YMagnetometer.FindMagnetometer()****YMagnetometer****yFindMagnetometer()yFindMagnetometer()**

Permet de retrouver un magnétomètre d'après un identifiant donné.

```
function yFindMagnetometer( ByVal func As String) As YMagnetometer
```

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() yFirstMagnetometer()yFirstMagnetometer()

**Y Magnetometer**

Commence l'énumération des magnétomètres accessibles par la librairie.

```
function yFirstMagnetometer( ) As YMagnetometer
```

Utiliser la fonction `YMagnetometer.nextMagnetometer()` pour itérer sur les autres magnétomètres.

**Retourne :**

un pointeur sur un objet `YMagnetometer`, correspondant au premier magnétomètre accessible en ligne, ou `null` si il n'y a pas de magnétomètres disponibles.

**magnetometer→calibrateFromPoints()**  
**magnetometer.calibrateFromPoints()****YMagnetometer**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→describe()magnetometer.describe()****YMagnetometer**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le magnétomètre (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**magnetometer→get\_advertisedValue()**  
**magnetometer→advertisedValue()**  
**magnetometer.get\_advertisedValue()**

**YMagnetometer**

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

function **get\_advertisedValue( ) As String**

**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), en mT, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule

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, en mT, sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule

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.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

---

**magnetometer→get\_errorType()**  
**magnetometer→errorType()**  
**magnetometer.get\_errorType()**

**YMagnetometer**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**magnetometer→functionDescriptor()**  
**magnetometer.get\_functionDescriptor()**

---

**YMagnetometer**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**magnetometer→get\_hardwareId()**  
**magnetometer→hardwareId()**  
**magnetometer.get\_hardwareId()**

**YMagnetometer**

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du magnétomètre (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le magnétomètre (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**magnetometer→get\_highestValue()**  
**magnetometer→highestValue()**  
**magnetometer.get\_highestValue()**

**YMagnetometer**

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**magnetometer→get\_logFrequency()**  
**magnetometer→logFrequency()**  
**magnetometer.get\_logFrequency()**

**YMagnetometer**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**magnetometer→get\_logicalName()**  
**magnetometer→logicalName()**  
**magnetometer.get\_logicalName()**

**YMagnetometer**

Retourne le nom logique du magnétomètre.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du magnétomètre.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**magnetometer→get\_lowestValue()**  
**magnetometer→lowestValue()**  
**magnetometer.get\_lowestValue()**

**YMagnetometer**

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**magnetometer→get\_module()**  
**magnetometer→module()**  
**magnetometer.get\_module()**

**YMagnetometer**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**magnetometer→get\_recordedData()**  
**magnetometer→recordedData()**  
**magnetometer.get\_recordedData()**

**YMagnetometer**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

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.

**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.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**magnetometer→get\_resolution()**  
**magnetometer→resolution()**  
**magnetometer.get\_resolution()**

---

**YMagnetometer**

Retourne la résolution des valeurs mesurées.

**function get\_resolution( ) As Double**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**magnetometer→get\_unit()****YMagnetometer****magnetometer→unit()magnetometer.get\_unit()**

Retourne l'unité dans laquelle le champ magnétique est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le champ magnétique est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**magnetometer→get(userData)**  
**magnetometer→userData()**  
**magnetometer.get(userData)**

**YMagnetometer**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**magnetometer→get\_xValue()****YMagnetometer****magnetometer→xValue()magnetometer.get\_xValue()**

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

```
function get_xValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la composante X du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_XVALUE\_INVALID**.

**magnetometer→get\_yValue()**

**YMagnetometer**

**magnetometer→yValue()magnetometer.get\_yValue()**

---

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

**function get\_yValue( ) As Double**

**Retourne :**

une valeur numérique représentant la composante Y du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_YVALUE_INVALID`.

**magnetometer→get\_zValue()****YMagnetometer****magnetometer→zValue()magnetometer.get\_zValue()**

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

```
function get_zValue( ) As Double
```

**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.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du magnétomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le magnétomètre est joignable, false sinon

**magnetometer→load()magnetometer.load()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

procedure **loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→nextMagnetometer()**  
**magnetometer.nextMagnetometer()**

**Y Magnetometer**

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

function **nextMagnetometer( ) As Y Magnetometer**

**Retourne :**

un pointeur sur un objet `Y Magnetometer` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**magnetometer→registerTimedReportCallback()**  
**magnetometer.registerTimedReportCallback()****YMagnetometer**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**magnetometer→set\_highestValue()**  
**magnetometer→setHighestValue()**  
**magnetometer.set\_highestValue()**

**YMagnetometer**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_logFrequency()**  
**magnetometer→setLogFrequency()**  
**magnetometer.set\_logFrequency()**

**YMagnetometer**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_logicalName()**  
**magnetometer→setLogicalName()**  
**magnetometer.set\_logicalName()**

**YMagnetometer**

Modifie le nom logique du magnétomètre.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du magnétomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_lowestValue()**  
**magnetometer→setLowestValue()**  
**magnetometer.set\_lowestValue()**

**YMagnetometer**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_reportFrequency()**  
**magnetometer→setReportFrequency()**  
**magnetometer.set\_reportFrequency()**

**YMagnetometer**

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**magnetometer→set\_resolution()**  
**magnetometer→setResolution()**  
**magnetometer.set\_resolution()**

**YMagnetometer**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

```
procedure set(userData) ( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.25. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Méthodes des objets YMeasure

#### **measure→get\_averageValue()**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

#### **measure→get\_maxValue()**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_minValue()**

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

#### **measure→get\_startTimeUTC()**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**measure→get\_averageValue()**  
**measure→averageValue()**  
**measure.get\_averageValue()**

**YMeasure**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

**function get\_averageValue( ) As Double**

**Retourne :**

un nombre décimal correspondant à la valeur moyenne observée.

**measure→get\_endTimeUTC()****YMeasure****measure→endTimeUTC()measure.get\_endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

```
function get_endTimeUTC( ) As Double
```

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclurent une fraction décimale.

**Retourne :**

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

**measure→get\_maxValue()**

**YMeasure**

**measure→maxValue()measure.get\_maxValue()**

---

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

function **get\_maxValue( ) As Double**

**Retourne :**

un nombre décimal correspondant à la plus grande valeur observée.

---

**measure→get\_minValue()****YMeasure****measure→minValue()measure.get\_minValue()**

---

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.function **get\_minValue( ) As Double****Retourne :**

un nombre décimal correspondant à la plus petite valeur observée.

---

<b>measure→getStartTimeUTC()</b>	<b>YMeasure</b>
<b>measure→startTimeUTC()</b>	
<b>measure.getStartTimeUTC()</b>	

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

**function getStartTimeUTC( ) As Double**

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.

## 3.26. 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→checkFirmware(path, onlynew)**

Test si le fichier byn est valid pour le module.

#### **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\_allSettings()**

Retourne tous les paramètres 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()**

### 3. Reference

Retourne la version du logiciel embarqué du module.
<b>module→get_hardwareId()</b> Retourne l'identifiant unique du module.
<b>module→get_icon2d()</b> 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_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→get(userVar)</b> Retourne la valeur entière précédemment stockée dans cet attribut.
<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> Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

**module→revertFromFlash()**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

**module→saveToFlash()**

Sauve les réglages courants dans la mémoire non volatile du module.

**module→set\_allSettings(settings)**

Restore tous les paramètres 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(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).

**module→set\_userVar(newval)**

Retourne la valeur entière précédemment stockée dans cet attribut.

**module→triggerFirmwareUpdate(secBeforeReboot)**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

**module→updateFirmware(path)**

Prepare une mise à jour de firmware du module.

**module→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YModule.FindModule()  
yFindModule()yFindModule()****YModule**

Permet de retrouver un module d'après son numéro de série ou son nom logique.

```
function yFindModule( ByVal func As String) As YModule
```

Cette fonction n'exige pas que le module soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YModule.isOnline()` pour tester si le module est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

`func` une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

**Retourne :**

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

**YModule.FirstModule()****YModule****yFirstModule()yFirstModule()**

Commence l'énumération des modules accessibles par la librairie.

```
function yFirstModule( ) As YModule
```

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→checkFirmware()|module.checkFirmware()****YModule**

Test si le fichié byn est valid pour le module.

```
function checkFirmware( ) As String
```

Cette methode est utile pour tester si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contiens plusieurs fichier byn. Dans ce cas cette methode retourne le path du fichier byn compatible le plus récent. Si le parametre onlynew est vrais les firmware équivalent ou plus ancien au firmware installé sont ignorés.

**Paramètres :**

**path** le path sur un fichier byn ou un répertoire contenant plusieurs fichier byn

**onlynew** retourne uniquement les fichier strictement plus récent

**Retourne :**

: le path du fichier byn a utiliser ou une chaine vide si aucun firmware plus récent est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

**module→describe()module.describe()****YModule**

Retourne un court texte décrivant le module.

```
function describe( ) As String
```

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

**Retourne :**

une chaîne de caractères décrivant le module

## module→download()module.download()

YModule

Télécharge le fichier choisi du module et retourne son contenu.

```
function download( ) As Byte
```

**Paramètres :**

**pathname** nom complet du fichier

**Retourne :**

le contenu du fichier chargé

En cas d'erreur, déclenche une exception ou retourne YAPI\_INVALID\_STRING.

**module→functionCount()module.functionCount()****YModule**

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

function **functionCount( )** As Integer

**Retourne :**

le nombre de fonctions sur le module

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→functionId()module.functionId()****YModule**

Retourne l'identifiant matériel de la *n*ième fonction du module.

```
function functionId( ByVal functionIndex As Integer) As String
```

**Paramètres :**

**functionIndex** l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

**Retourne :**

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

**module→functionName()module.functionName()****YModule**

Retourne le nom logique de la *n*ième fonction du module.

```
function functionName( ByVal functionIndex As Integer) As String
```

**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.

```
function functionValue( ByVal functionIndex As Integer) As String
```

**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→get_allSettings()</b>	<b>YModule</b>
<b>module→allSettings()module.get_allSettings()</b>	

---

Retourne tous les paramètres du module.

function **get\_allSettings( ) As Byte**

Utile pour sauvgarder les noms logiques et les calibrations du module.

**Retourne :**

un buffer binaire avec tous les paramètres En cas d'erreur, déclenche une exception ou retourne YAPI\_INVALID\_STRING.

**module→get\_beacon()**  
**module→beacon()module.get\_beacon()**

---

**YModule**

Retourne l'état de la balise de localisation.

**function get\_beacon( ) As Integer**

**Retourne :**

soit Y\_BEACON\_OFF, soit Y\_BEACON\_ON, selon l'état de la balise de localisation

En cas d'erreur, déclenche une exception ou retourne Y\_BEACON\_INVALID.

**module→getErrorMessage()****YModule****module→errorMessage()module.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

**module→get\_errorType()** **YModule**  
**module→errorType()module.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
function get_errorType( ) As YRETCODE
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

**module→get\_firmwareRelease()**  
**module→firmwareRelease()**  
**module.get\_firmwareRelease()**

**YModule**

Retourne la version du logiciel embarqué du module.

```
function get_firmwareRelease( ) As String
```

**Retourne :**

une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne Y\_FIRMWARERELEASE\_INVALID.

**module→get\_hardwareId()**

**YModule**

**module→hardwareId()module.get\_hardwareId()**

---

Retourne l'identifiant unique du module.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

**Retourne :**

une chaîne de caractères identifiant la fonction

---

<b>module→get_icon2d()</b>	<b>YModule</b>
<b>module→icon2d()module.get_icon2d()</b>	

---

Retourne l'icône du module.

```
function get_icon2d( ) As Byte
```

L'icone est au format PNG et a une taille maximale de 1536 octets.

**Retourne :**

un buffer binaire contenant l'icone, au format png. En cas d'erreur, déclenche une exception ou retourne YAPI\_INVALID\_STRING.

**module→get\_lastLogs()**  
**module→lastLogs()module.get\_lastLogs()**

---

**YModule**

Retourne une chaîne de caractère contenant les derniers logs du module.

**function get\_lastLogs( ) As String**

Cette méthode retourne les derniers logs qui sont encore stocké dans le module.

**Retourne :**

une chaîne de caractère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne YAPI\_INVALID\_STRING.

---

<b>module→get_logicalName()</b>	<b>YModule</b>
<b>module→logicalName()module.get_logicalName()</b>	

---

Retourne le nom logique du module.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**module→get\_luminosity()**

**YModule**

**module→luminosity()module.get\_luminosity()**

---

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

**function get\_luminosity( ) As Integer**

**Retourne :**

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne Y\_LUMINOSITY\_INVALID.

**module→get\_persistentSettings()**  
**module→persistentSettings()**  
**module.get\_persistentSettings()**

**YModule**

Retourne l'état courant des réglages persistents du module.

```
function get_persistentSettings( ) As Integer
```

**Retourne :**

une valeur parmi Y\_PERSISTENTSETTINGS\_LOADED, Y\_PERSISTENTSETTINGS\_SAVED et Y\_PERSISTENTSETTINGS\_MODIFIED représentant l'état courant des réglages persistents du module

En cas d'erreur, déclenche une exception ou retourne Y\_PERSISTENTSETTINGS\_INVALID.

**module→get\_productId()**  
**module→productId()module.get\_productId()**

---

**YModule**

Retourne l'identifiant USB du module, préprogrammé en usine.

**function get\_productId( ) As Integer**

**Retourne :**

un entier représentant l'identifiant USB du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTID\_INVALID.

---

<b>module→get_productName()</b>	<b>YModule</b>
<b>module→productName()module.get_productName()</b>	

---

Retourne le nom commercial du module, préprogrammé en usine.

```
function get_productName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTNAME\_INVALID.

**module→get\_productRelease()**  
**module→productRelease()**  
**module.get\_productRelease()**

---

**YModule**

Retourne le numéro de version matériel du module, préprogrammé en usine.

**function get\_productRelease( ) As Integer**

**Retourne :**

un entier représentant le numéro de version matériel du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_PRODUCTRELEASE\_INVALID.

**module→get\_rebootCountdown()**  
**module→rebootCountdown()**  
**module.get\_rebootCountdown()**

**YModule**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

function **get\_rebootCountdown( ) As Integer**

**Retourne :**

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

En cas d'erreur, déclenche une exception ou retourne **Y\_REBOOTCOUNTDOWN\_INVALID**.

**module→get\_serialNumber()** **YModule**  
**module→serialNumber()module.get\_serialNumber()**

---

Retourne le numéro de série du module, préprogrammé en usine.

**function get\_serialNumber( ) As String**

**Retourne :**

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y\_SERIALNUMBER\_INVALID.

---

<b>module-&gt;get_upTime()</b>	<b>YModule</b>
<b>module-&gt;upTime()module.get_upTime()</b>	

---

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

```
function get_upTime( ) As Long
```

**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\_usbCurrent()**

**YModule**

**module→usbCurrent()module.get\_usbCurrent()**

---

Retourne le courant consommé par le module sur le bus USB, en milliampères.

**function get\_usbCurrent( ) As Integer**

**Retourne :**

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

En cas d'erreur, déclenche une exception ou retourne Y\_USBCURRENT\_INVALID.

**module→get(userData)****YModule****module→userData()module.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

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→get\_userVar()**  
**module→userVar()module.get\_userVar()**

---

**YModule**

Retourne la valeur entière précédemment stockée dans cet attribut.

**function get\_userVar( ) As Integer**

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

**Retourne :**

un entier représentant la valeur entière précédemment stockée dans cet attribut

En cas d'erreur, déclenche une exception ou retourne Y\_USERVAR\_INVALID.

**module→isOnline()module.isOnline()****YModule**

Vérifie si le module est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le module est joignable, false sinon

**module→load()module.load()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→nextModule()|module.nextModule()****YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

```
function nextModule() As YModule
```

**Retourne :**

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**module→reboot()module.reboot()****YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
function reboot( ) As Integer
```

**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**

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

```
function registerLogCallback( ByVal callback As YModuleLogCallback) As Integer
```

Utile pour débugger le fonctionnement d'un module Yoctopuce.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet module qui a produit un log, un chaîne de caractère qui contiens le log

**module→revertFromFlash()**  
**module.revertFromFlash()**

---

**YModule**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

```
function revertFromFlash( ) As Integer
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→saveToFlash()module.saveToFlash()****YModule**

Sauve les réglages courants dans la mémoire non volatile du module.

```
function saveToFlash( ) As Integer
```

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\_allSettings()** YModule  
**module→setAllSettings()module.set\_allSettings()**

---

Restore tous les paramètres du module.

**procedure set\_allSettings( )**

Utile pour restorer les noms logiques et les calibrations du module depuis un sauvegarde.

**Paramètres :**

**settings** un buffer binaire avec tous les paramètres

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>module→set_beacon()</b>	<b>YModule</b>
<b>module→setBeacon()module.set_beacon()</b>	

---

Allume ou éteint la balise de localisation du module.

```
function set_beacon( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_BEACON\_OFF, soit Y\_BEACON\_ON

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**module→set\_logicalName()** YModule  
**module→setLogicalName()****module.set\_logicalName()**

---

Change le nom logique du module.

```
function set_logicalName( ByVal newval As String) As Integer
```

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()</b>	<b>module.set_luminosity()</b>

---

Modifie la luminosité des leds informatives du module.

```
function set_luminosity( ByVal newval As Integer) As Integer
```

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(userData)** **YModule**  
**module→setUserData()module.set(userData)**

---

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
procedure set(userData( ByVal data As Object)
```

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→set\_userVar()****YModule****module→setUserVar()module.set\_userVar()**

Retourne la valeur entière précédemment stockée dans cet attribut.

```
function set_userVar( ByVal newval As Integer) As Integer
```

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

**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.

**module→triggerFirmwareUpdate()**  
**module.triggerFirmwareUpdate()**

**YModule**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

function **triggerFirmwareUpdate( ) As Integer**

**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→updateFirmware()module.updateFirmware()****YModule**

Prepare une mise à jour de firmware du module.

```
function updateFirmware( ) As YFirmwareUpdate
```

Cette methode un object YFirmwareUpdate qui est utilisé pour mettre à jour le firmware du module.

**Paramètres :**

**path** le path sur un fichier byn

**Retourne :**

: Un object YFirmwareUpdate

## 3.27. Interface de la fonction Motor

La librairie de programmation yoctopuce permet de piloter la puissance envoyée au moteur pour le faire tourner aussi bien dans un sens que dans l'autre, mais aussi de piloter des accélérations linéaires: le moteur accélère alors tout seul sans que vous vous ayez à vous en occuper. La librairie permet aussi de freiner le moteur: cela est réalisé en court-circuitant les pôles du moteur, ce qui le transforme en frein électro-magnétique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_motor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YMotor = yoctolib.YMotor;
php require_once('yocto_motor.php');
cpp #include "yocto_motor.h"
m #import "yocto_motor.h"
pas uses yocto_motor;
vb yocto_motor.vb
cs yocto_motor.cs
java import com.yoctopuce.YoctoAPI.YMotor;
py from yocto_motor import *

```

### Fonction globales

#### yFindMotor(func)

Permet de retrouver un moteur d'après un identifiant donné.

#### yFirstMotor()

Commence l'énumération des moteur accessibles par la librairie.

### Méthodes des objets YMotor

#### **motor→brakingForceMove(targetPower, delay)**

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

#### **motor→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **motor→drivingForceMove(targetPower, delay)**

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

#### **motor→get\_advertisedValue()**

Retourne la valeur courante du moteur (pas plus de 6 caractères).

#### **motor→get\_brakingForce()**

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

#### **motor→get\_cutOffVoltage()**

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

#### **motor→get\_drivingForce()**

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

#### **motor→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

#### **motor→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

#### **motor→get\_failSafeTimeout()**

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

#### **motor→get\_frequency()**

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

#### **motor→get\_friendlyName()**

Retourne un identifiant global du moteur au format NOM\_MODULE . NOM\_FONCTION.

#### **motor→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **motor→get\_functionId()**

Retourne l'identifiant matériel du moteur, sans référence au module.

#### **motor→get\_hardwareId()**

Retourne l'identifiant matériel unique du moteur au format SERIAL . FUNCTIONID.

#### **motor→get\_logicalName()**

Retourne le nom logique du moteur.

#### **motor→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **motor→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **motor→get\_motorStatus()**

Retourne l'état du contrôleur de moteur.

#### **motor→get\_overCurrentLimit()**

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

#### **motor→get\_starterTime()**

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

#### **motor→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **motor→isOnline()**

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

#### **motor→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

#### **motor→keepALive()**

Réarme la sécurité failsafe du contrôleur.

#### **motor→load(msValidity)**

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

#### **motor→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

#### **motor→nextMotor()**

Continue l'énumération des moteur commencée à l'aide de yFirstMotor( ).

#### **motor→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **motor→resetStatus()**

Réinitialise l'état du contrôleur à IDLE.

#### **motor→set\_brakingForce(newval)**

### 3. Reference

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

#### **motor→set\_cutOffVoltage(newval)**

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

#### **motor→set\_drivingForce(newval)**

Modifie immédiatement la puissance envoyée au moteur.

#### **motor→set\_failSafeTimeout(newval)**

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

#### **motor→set\_frequency(newval)**

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

#### **motor→set\_logicalName(newval)**

Modifie le nom logique du moteur.

#### **motor→set\_overCurrentLimit(newval)**

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

#### **motor→set\_starterTime(newval)**

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

#### **motor→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).

#### **motor→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.

## YMotor.FindMotor() yFindMotor()yFindMotor()

YMotor

Permet de retrouver un moteur d'après un identifiant donné.

```
function yFindMotor( ByVal func As String) As YMotor
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMotor.isOnline()` pour tester si le moteur 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 moteur sans ambiguïté

### Retourne :

un objet de classe `YMotor` qui permet ensuite de contrôler le moteur.

## YMotor.FirstMotor() yFirstMotor()yFirstMotor()

YMotor

Commence l'énumération des moteur accessibles par la librairie.

```
function yFirstMotor( ) As YMotor
```

Utiliser la fonction `YMotor.nextMotor()` pour itérer sur les autres moteur.

**Retourne :**

un pointeur sur un objet `YMotor`, correspondant au premier moteur accessible en ligne, ou `null` si il n'y a pas de moteur disponibles.

**motor→brakingForceMove()**  
**motor.brakingForceMove()****YMotor**

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

function **brakingForceMove( ) As Integer**

**Paramètres :**

**targetPower** force de freinage finale, en pourcentage

**delay** durée (en ms) sur laquelle le changement de puissance sera effectué

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→describe()motor.describe()****YMotor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le moteur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**motor→drivingForceMove()**  
**motor.drivingForceMove()****YMotor**

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

```
function drivingForceMove( ) As Integer
```

**Paramètres :**

**targetPower** puissance finale désirée, en pourcentage de -100% à +100%

**delay** durée (en ms) sur laquelle le changement de puissance sera effectué

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→get\_advertisedValue()**  
**motor→advertisedValue()**  
**motor.get\_advertisedValue()**

---

**YMotor**

Retourne la valeur courante du moteur (pas plus de 6 caractères).

**function get\_advertisedValue( ) As String**

**Retourne :**

une chaîne de caractères représentant la valeur courante du moteur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**motor→get\_brakingForce()****YMotor****motor→brakingForce()motor.get\_brakingForce()**

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

```
function get_brakingForce( ) As Double
```

La valeur 0 correspond ne pas freiner (moteur en roue libre).

**Retourne :**

une valeur numérique représentant la force de freinage appliquée au moteur, sous forme de pourcentage

En cas d'erreur, déclenche une exception ou retourne Y\_BRAKINGFORCE\_INVALID.

**motor→get\_cutOffVoltage()** YMotor  
**motor→cutOffVoltage()motor.get\_cutOffVoltage()**

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

**function get\_cutOffVoltage( ) As Double**

Ce réglage permet d'éviter d'endommager un accumulateur en continuant à l'utiliser une fois "vide".

**Retourne :**

une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

En cas d'erreur, déclenche une exception ou retourne Y\_CUTOFFVOLTAGE\_INVALID.

**motor→get\_drivingForce()****YMotor****motor→drivingForce()motor.get\_drivingForce()**

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

```
function get_drivingForce( ) As Double
```

**Retourne :**

une valeur numérique représentant la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%

En cas d'erreur, déclenche une exception ou retourne Y\_DRIVINGFORCE\_INVALID.

**motor→getErrorMessage()**

**YMotor**

**motor→errorMessage()motor.getErrorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

**function getErrorMessage( ) As String**

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 moteur.

**motor→get\_errorType()****YMotor****motor→errorType()motor.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

```
function get_errorType( ) As YRETCODE
```

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 moteur.

**motor→get\_failSafeTimeout()** YMotor  
**motor→failSafeTimeout()motor.get\_failSafeTimeout()**

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

**function get\_failSafeTimeout( ) As Integer**

Passé ce délai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

**Retourne :**

un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

En cas d'erreur, déclenche une exception ou retourne Y\_FAILSAFETIMEOUT\_INVALID.

**motor→get\_frequency()****YMotor****motor→frequency()motor.get\_frequency()**

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

```
function get_frequency( ) As Double
```

**Retourne :**

une valeur numérique représentant la fréquence du signal PWM utilisé pour contrôler le moteur

En cas d'erreur, déclenche une exception ou retourne Y\_FREQUENCY\_INVALID.

**motor→get\_functionDescriptor()**  
**motor→functionDescriptor()**  
**motor.get\_functionDescriptor()**

**YMotor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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

---

**motor→get\_functionId()****YMotor****motor→functionId()motor.get\_functionId()**

---

Retourne l'identifiant matériel du moteur, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.**Retourne :**une chaîne de caractères identifiant le moteur (ex: `relay1`)En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**motor→get\_hardwareId()**

**YMotor**

**motor→hardwareId()motor.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du moteur au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moteur (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le moteur (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

**motor→get\_logicalName()**  
**motor→logicalName()motor.get\_logicalName()**

---

**YMotor**

Retourne le nom logique du moteur.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du moteur.

En cas d'erreur, déclenche une exception ou retourne **Y\_LOGICALNAME\_INVALID**.

**motor→get\_module()**

**YMotor**

**motor→module()motor.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

**motor→get\_motorStatus()**  
**motor→motorStatus()motor.get\_motorStatus()****YMotor**

Retourne l'état du contrôleur de moteur.

```
function get_motorStatus( ) As Integer
```

Les états possibles sont: IDLE si le moteur est à l'arrêt/en roue libre, prêt à démarrer; FORWD si le contrôleur fait tourner le moteur en marche avant; BACKWD si le contrôleur fait tourner le moteur en marche arrière; BRAKE si le contrôleur est en train de freiner; LOVOLT si le contrôleur a détecté une tension trop basse; HICURR si le contrôleur a détecté une surconsommation; HIHEAT si le contrôleur a détecté une surchauffe; FAILSF si le contrôleur est passé en protection failsafe.

Si le contrôleur est en erreur (LOVOLT, HICURR, HIHEAT,FAILSF), il doit être explicitement réinitialisé avec la fonction `resetStatus`.

**Retourne :**

```
une valeur parmi Y_MOTORSTATUS_IDLE, Y_MOTORSTATUS_BRAKE,  
Y_MOTORSTATUS_FORWD, Y_MOTORSTATUS_BACKWD, Y_MOTORSTATUS_LOVOLT,  
Y_MOTORSTATUS_HICURR, Y_MOTORSTATUS_HIHEAT et Y_MOTORSTATUS_FAILSF  
représentant l'état du contrôleur de moteur
```

En cas d'erreur, déclenche une exception ou retourne Y\_MOTORSTATUS\_INVALID.

**motor→get\_overCurrentLimit()**  
**motor→overCurrentLimit()**  
**motor.get\_overCurrentLimit()**

**YMotor**

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

**function get\_overCurrentLimit( ) As Integer**

Une valeur nulle signifie qu'aucune limite n'est définie.

**Retourne :**

un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

En cas d'erreur, déclenche une exception ou retourne Y\_OVERCURRENTLIMIT\_INVALID.

**motor→get\_starterTime()****YMotor****motor→starterTime()motor.get\_starterTime()**

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

```
function get_starterTime( ) As Integer
```

**Retourne :**

un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

En cas d'erreur, déclenche une exception ou retourne Y\_STARTERTIME\_INVALID.

**motor→get(userData)**

**YMotor**

**motor→userData()motor.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**function get(userData) As Object**

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.

**motor→isOnline()motor.isOnline()****YMotor**

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du moteur sont valides au moment de l'appel, le module est considéré joignable. 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 moteur est joignable, false sinon

**motor→keepALive()motor.keepALive()****YMotor**

Réarme la sécurité failsafe du contrôleur.

```
function keepALive( ) As Integer
```

Lorsque le moteur est en marche et que la sécurité failsafe est activée, cette fonction doit être appelée périodiquement pour confirmer le bon fonctionnement du processus de contrôle. A défaut, le moteur s'arrêtera automatiquement au bout du temps prévu. Notez que l'appel à une fonction de type *set* du moteur réarme aussi la sécurité failsafe.

**motor→load()motor.load()****YMotor**

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

## **motor→nextMotor()motor.nextMotor()**

**YMotor**

---

Continue l'énumération des moteur commencée à l'aide de `yFirstMotor()`.

```
function nextMotor( ) As YMotor
```

**Retourne :**

un pointeur sur un objet `YMotor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**motor→registerValueCallback()  
motor.registerValueCallback()****YMotor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

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.

## **motor→resetStatus()motor.resetStatus()**

**YMotor**

Réinitialise l'état du contrôleur à IDLE.

```
function resetStatus( ) As Integer
```

Cette fonction doit être explicitement appelée après toute condition d'erreur pour permettre au contrôleur de repartir.

**motor→set\_brakingForce()****YMotor****motor→setBrakingForce()motor.set\_brakingForce()**

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

```
function set_brakingForce( ByVal newval As Double) As Integer
```

La valeur 0 correspond à ne pas freiner (moteur en roue libre). Lorsque la force de freinage est changée, la puissance de traction est remise à zéro.

**Paramètres :**

**newval** une valeur numérique représentant immédiatement la force de freinage appliquée au moteur (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.

---

<b>motor→set_cutOffVoltage()</b>	<b>YMotor</b>
<b>motor→setCutOffVoltage()motor.set_cutOffVoltage()</b>	

---

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

```
function set_cutOffVoltage( ByVal newval As Double) As Integer
```

Ce réglage permet d'éviter d'endommager un accumulateur en continuant à l'utiliser une fois "vide". Attention, quel que soit le réglage du cutoff, le variateur passera en erreur si l'alimentation passe (même brièvement) en dessous de 3V.

**Paramètres :**

**newval** une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

**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>motor→set_drivingForce()</b>	<b>YMotor</b>
<b>motor→setDrivingForce()motor.set_drivingForce()</b>	

---

Modifie immédiatement la puissance envoyée au moteur.

```
function set_drivingForce( ByVal newval As Double) As Integer
```

La valeur est donnée en pourcentage de -100% à +100%. Si vous voulez ménager votre mécanique et éviter d'induire des consommations excessives qui pourraient dépasser les capacités du contrôleur, évitez les changements de régime trop brusques. Par exemple, passer brutalement de marche avant à marche arrière est une très mauvaise idée. A chaque fois que la puissance envoyée au moteur est changée, le freinage est remis à zéro.

**Paramètres :**

**newval** une valeur numérique représentant immédiatement la puissance envoyée au moteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→set\_failSafeTimeout()**  
**motor→setFailSafeTimeout()**  
**motor.set\_failSafeTimeout()**

YMotor

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

```
function set_failSafeTimeout( ByVal newval As Integer) As Integer
```

Passé ce délai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

**Paramètres :**

**newval** un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→set\_frequency()****YMotor****motor→setFrequency()motor.set\_frequency()**

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

```
function set_frequency( ByVal newval As Double) As Integer
```

Une fréquence basse est généralement plus efficace (les composant chauffent moins et le moteur démarre plus facilement), mais un bruit audible peut être généré. Une fréquence élevée peut réduire le bruit, mais il y a plus d'énergie perdue en chaleur.

**Paramètres :**

**newval** une valeur numérique représentant la fréquence du signal PWM utilisée pour contrôler le moteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→set\_logicalName()** **YMotor**  
**motor→setLogicalName()motor.set\_logicalName()**

Modifie le nom logique du moteur.

```
function set_logicalName( ByVal newval As String) As Integer
```

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 moteur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→set\_overCurrentLimit()**  
**motor→setOverCurrentLimit()**  
**motor.set\_overCurrentLimit()**

YMotor

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

function **set\_overCurrentLimit( ByVal newval As Integer) As Integer**

Une valeur nulle signifie qu'aucune limite n'est définie. Attention, quel que soit le réglage choisi, le variateur passera en erreur si le courant passe, même brièvement, en dessus de 32A.

**Paramètres :**

**newval** un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en 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.

**motor→set\_starterTime()** YMotor  
**motor→setStarterTime()motor.set\_starterTime()**

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

```
function set_starterTime( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**motor→set(userData)****YMotor****motor→setUserData()motor.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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.28. Interface de la fonction Network

Les objets YNetwork permettent de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_network.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YNetwork = yoctolib.YNetwork;
php	require_once('yocto_network.php');
cpp	#include "yocto_network.h"
m	#import "yocto_network.h"
pas	uses yocto_network;
vb	yocto_network.vb
cs	yocto_network.cs
java	import com.yoctopuce.YoctoAPI.YNetwork;
py	from yocto_network import *

### Fonction globales

#### yFindNetwork(func)

Permet de retrouver une interface réseau d'après un identifiant donné.

#### yFirstNetwork()

Commence l'énumération des interfaces réseau accessibles par la librairie.

### Méthodes des objets YNetwork

#### network→callbackLogin(username, password)

Contacte le callback de notification et sauvegarde un laissez-passer pour s'y connecter.

#### network→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### network→get\_adminPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

#### network→get\_advertisedValue()

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

#### network→get\_callbackCredentials()

Retourne une version hashée du laissez-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

#### network→get\_callbackEncoding()

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

#### network→get\_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

#### network→get\_callbackMethod()

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

#### network→get\_callbackMinDelay()

Retourne l'attente minimale entre deux notifications par callback, en secondes.

#### network→get\_callbackUrl()

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

#### network→get\_discoverable()

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

#### **network→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

#### **network→get\_friendlyName()**

Retourne un identifiant global de l'interface réseau au format NOM\_MODULE . NOM\_FONCTION.

#### **network→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **network→get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

#### **network→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL . FUNCTIONID.

#### **network→get\_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

#### **network→get\_logicalName()**

Retourne le nom logique de l'interface réseau.

#### **network→get\_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

#### **network→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **network→get\_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

#### **network→get\_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

#### **network→get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

#### **network→get\_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

#### **network→get\_secondaryDNS()**

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

#### **network→get\_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

#### **network→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **network→get\_userPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

#### **network→get\_wwwWatchdogDelay()**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

#### **network→isOnline()**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

### 3. Reference

<b>network→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.
<b>network→load(msValidity)</b>
Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.
<b>network→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.
<b>network→nextNetwork()</b>
Continue l'énumération des interfaces réseau commencée à l'aide de <code>yFirstNetwork()</code> .
<b>network→ping(host)</b>
Ping <code>str_host</code> pour vérifier la connexion réseau.
<b>network→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>network→set_adminPassword(newval)</b>
Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.
<b>network→set_callbackCredentials(newval)</b>
Modifie le laisser-passer pour se connecter à l'adresse de callback.
<b>network→set_callbackEncoding(newval)</b>
Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.
<b>network→set_callbackMaxDelay(newval)</b>
Modifie l'attente maximale entre deux notifications par callback, en secondes.
<b>network→set_callbackMethod(newval)</b>
Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.
<b>network→set_callbackMinDelay(newval)</b>
Modifie l'attente minimale entre deux notifications par callback, en secondes.
<b>network→set_callbackUrl(newval)</b>
Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.
<b>network→set_discoverable(newval)</b>
Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).
<b>network→set_logicalName(newval)</b>
Modifie le nom logique de l'interface réseau.
<b>network→set_primaryDNS(newval)</b>
Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.
<b>network→set_secondaryDNS(newval)</b>
Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.
<b>network→set_userData(data)</b>
Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get(userData)</code> .
<b>network→set_userPassword(newval)</b>
Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.
<b>network→set_wwwWatchdogDelay(newval)</b>
Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.
<b>network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)</b>

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

**network→useStaticIP(ipAddress, subnetMaskLen, router)**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

**network→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YNetwork.FindNetwork() yFindNetwork()yFindNetwork()

**YNetwork**

Permet de retrouver une interface réseau d'après un identifiant donné.

```
function yFindNetwork( ByVal func As String) As YNetwork
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YNetwork.isOnline()` pour tester si l'interface réseau est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'interface réseau sans ambiguïté

### Retourne :

un objet de classe `YNetwork` qui permet ensuite de contrôler l'interface réseau.

**YNetwork.FirstNetwork()****YNetwork****yFirstNetwork()yFirstNetwork()**

Commence l'énumération des interfaces réseau accessibles par la librairie.

```
function yFirstNetwork( ) As YNetwork
```

Utiliser la fonction `YNetwork.nextNetwork()` pour itérer sur les autres interfaces réseau.

**Retourne :**

un pointeur sur un objet `YNetwork`, correspondant à la première interface réseau accessible en ligne, ou `null` si il n'y a pas de interfaces réseau disponibles.

**network→callbackLogin()network.callbackLogin()****YNetwork**

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

```
function callbackLogin( ByVal username As String,  
                      ByVal password As String) As Integer
```

Le mot de passe ne sera pas stocké dans le module, mais seulement une version hashée non réversible. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**username** nom d'utilisateur pour s'identifier au callback

**password** mot de passe pour s'identifier au callback

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→describe()network.describe()****YNetwork**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant l'interface réseau (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**network→get\_adminPassword()**  
**network→adminPassword()**  
**network.get\_adminPassword()**

**YNetwork**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

```
function get_adminPassword( ) As String
```

**Retourne :**

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_ADMINPASSWORD\_INVALID.

**network→get\_advertisedValue()**  
**network→advertisedValue()**  
**network.get\_advertisedValue()**

**YNetwork**

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

function **get\_advertisedValue( ) As String**

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'interface réseau (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**network→get\_callbackCredentials()**  
**network→callbackCredentials()**  
**network.get\_callbackCredentials()**

YNetwork

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

function **get\_callbackCredentials( ) As String**

**Retourne :**

une chaîne de caractères représentant une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKCREDENTIALS\_INVALID.

**network→get\_callbackEncoding()**  
**network→callbackEncoding()**  
**network.get\_callbackEncoding()**

**YNetwork**

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
function get_callbackEncoding( ) As Integer
```

**Retourne :**

une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON,  
Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et  
Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs  
notifiées par callback

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKENCODING\_INVALID.

**network→get\_callbackMaxDelay()**  
**network→callbackMaxDelay()**  
**network.get\_callbackMaxDelay()**

---

**YNetwork**

Retourne l'attente maximale entre deux notifications par callback, en secondes.

```
function get_callbackMaxDelay( ) As Integer
```

**Retourne :**

un entier représentant l'attente maximale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMAXDELAY\_INVALID.

**network→get\_callbackMethod()**  
**network→callbackMethod()**  
**network.get\_callbackMethod()**

**YNetwork**

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
function get_callbackMethod( ) As Integer
```

**Retourne :**

une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMETHOD\_INVALID.

**network→get\_callbackMinDelay()**  
**network→callbackMinDelay()**  
**network.get\_callbackMinDelay()**

---

**YNetwork**

Retourne l'attente minimale entre deux notifications par callback, en secondes.

**function get\_callbackMinDelay( ) As Integer**

**Retourne :**

un entier représentant l'attente minimale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKMINDELAY\_INVALID.

**network→get\_callbackUrl()****YNetwork****network→callbackUrl()network.get\_callbackUrl()**

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
function get_callbackUrl( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

En cas d'erreur, déclenche une exception ou retourne Y\_CALLBACKURL\_INVALID.

**network→get\_discoverable()**

**YNetwork**

**network→discoverable()network.get\_discoverable()**

---

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

**function get\_Discoverable( ) As Integer**

**Retourne :**

soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne Y\_DISCOVERABLE\_INVALID.

**network→get\_errorMessage()**  
**network→errorMessage()**  
**network.get\_errorMessage()****YNetwork**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

**network→get\_errorType()**

**YNetwork**

**network→errorType()network.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**network→functionDescriptor()**  
**network.get\_functionDescriptor()**

**YNetwork**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**network→get\_functionId()**

**YNetwork**

**network→functionId()network.get\_functionId()**

---

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

**function get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**network→get\_hardwareId()****YNetwork****network→hardwareId()network.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL.FUNCTIONID.

function **get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'interface réseau (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**network→get\_ipAddress()**

**YNetwork**

**network→ipAddress()network.get\_ipAddress()**

---

Retourne l'adresse IP utilisée par le module Yoctopuce.

```
function get_ipAddress( ) As String
```

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.

---

<b>network→get_logicalName()</b>	<b>YNetwork</b>
<b>network→logicalName()network.get_logicalName()</b>	

---

Retourne le nom logique de l'interface réseau.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**network→get\_macAddress()** YNetwork  
**network→macAddress()network.get\_macAddress()**

---

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

```
function get_macAddress( ) As String
```

L'adresse MAC est aussi présente sur un autocollant sur le module, représentée en chiffres et en code-barres.

**Retourne :**

une chaîne de caractères représentant l'adresse MAC de l'interface réseau, unique pour chaque module

En cas d'erreur, déclenche une exception ou retourne Y\_MACADDRESS\_INVALID.

**network→get\_module()****YNetwork****network→module()network.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**network→get\_poeCurrent()**

**YNetwork**

**network→poeCurrent()network.get\_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

```
function get_poeCurrent( ) As Integer
```

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

**Retourne :**

un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne Y\_POECURRENT\_INVALID.

---

<b>network→get_primaryDNS()</b>	<b>YNetwork</b>
<b>network→primaryDNS()network.get_primaryDNS()</b>	

---

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
function get_primaryDNS( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_PRIMARYDNS\_INVALID.

**network→get\_readiness()****YNetwork****network→readiness()network.get\_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

```
function get_readiness( ) As Integer
```

Le niveau zéro (DOWN\_0) signifie qu'aucun support réseau matériel n'a été détecté. Soit il n'y a pas de signal sur le câble réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE\_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme la l'existence du SSID configuré. Le niveau 2 (LINK\_2) est atteint lorsque le support matériel du réseau est fonctionnel. Pour une connection réseau filaire, le niveau 2 signifie que le câble est connecté aux deux bouts. Pour une connection à un point d'accès réseau sans fil, il démontre que les paramètres de sécurités configurés sont corrects. Pour une connection sans fil en mode ad-hoc, cela signifie qu'il y a au moins un partenaire sur le réseau ad-hoc. Le niveau 3 (DHCP\_3) est atteint lorsque qu'une adresse IP a été obtenue par DHCP. Le niveau 4 (DNS\_4) est atteint lorsqu'un serveur DNS est joignable par le réseau. Le niveau 5 (WWW\_5) est atteint lorsque la connectivité globale à internet est avérée par l'obtention de l'heure courante sur une serveur NTP.

**Retourne :**

une valeur parmi Y\_READINESS\_DOWN, Y\_READINESS\_EXISTS, Y\_READINESS\_LINKED, Y\_READINESS\_LAN\_OK et Y\_READINESS\_WWW\_OK représentant l'état de fonctionnement atteint par l'interface réseau

En cas d'erreur, déclenche une exception ou retourne Y\_READINESS\_INVALID.

**network→get\_router()****YNetwork****network→router()network.get\_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

```
function get_router( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*)

En cas d'erreur, déclenche une exception ou retourne Y\_ROUTER\_INVALID.

**network→get\_secondaryDNS()**  
**network→secondaryDNS()**  
**network.get\_secondaryDNS()**

**YNetwork**

---

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

```
function get_secondaryDNS( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y\_SECONDARYDNS\_INVALID.

---

<b>network→get_subnetMask()</b>	<b>YNetwork</b>
<b>network→subnetMask()network.get_subnetMask()</b>	

---

Retourne le masque de sous-réseau utilisé par le module.

```
function get_subnetMask( ) As String
```

**Retourne :**

une chaîne de caractères représentant le masque de sous-réseau utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_SUBNETMASK\_INVALID.

**network→get(userData)**

**YNetwork**

**network→userData()network.get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**function get(userData) As Object**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**network→get\_userPassword()**  
**network→userPassword()**  
**network.get\_userPassword()**

**YNetwork**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

function **get\_userPassword( ) As String**

**Retourne :**

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne **Y\_USERPASSWORD\_INVALID**.

**network→get\_wwwWatchdogDelay()**  
**network→wwwWatchdogDelay()**  
**network.get\_wwwWatchdogDelay()**

**YNetwork**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

**function get\_wwwWatchdogDelay( ) As Integer**

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW.

**Retourne :**

un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

En cas d'erreur, déclenche une exception ou retourne Y\_WWWWATCHDOGDELAY\_INVALID.

**network→isOnline()network.isOnline()****YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache de l'interface réseau sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'interface réseau est joignable, false sinon

**network→load()|network.load()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→nextNetwork()network.nextNetwork()****YNetwork**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

```
function nextNetwork( ) As YNetwork
```

**Retourne :**

un pointeur sur un objet `YNetwork` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**network→ping()network.ping()****YNetwork**

Ping str\_host pour vérifier la connexion réseau.

```
function ping( ) As String
```

Envoie quatre requêtes ICMP ECHO\_RESPONER à la cible str\_host depuis le module. Cette méthode retourne une chaîne de caractères avec le résultat des 4 requêtes ICMP ECHO\_RESPONSE.

**Paramètres :**

**host** le nom d'hôte ou l'adresse IP de la cible

**Retourne :**

une chaîne de caractères contenant le résultat du ping.

**network→registerValueCallback()**  
**network.registerValueCallback()****YNetwork**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**network→set\_adminPassword()**  
**network→setAdminPassword()**  
**network.set\_adminPassword()**

**YNetwork**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

function **set\_adminPassword( ByVal newval As String) As Integer**

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackCredentials()**  
**network→setCallbackCredentials()**  
**network.set\_callbackCredentials()**

**YNetwork**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

```
function set_callbackCredentials( ByVal newval As String) As Integer
```

Le laisser-passer doit être fourni tel que retourné par la fonction `get_callbackCredentials`, sous la forme `username:hash`. La valeur du hash dépend de la méthode d'autorisation implémentée par le callback. Pour une autorisation de type Basic, le hash est le MD5 de la chaîne `username:password`. Pour une autorisation de type Digest, le hash est le MD5 de la chaîne `username:realm:password`. Pour une utilisation simplifiée, utilisez la fonction `callbackLogin`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le laisser-passer pour se connecter à l'adresse de callback

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackEncoding()**  
**network→setCallbackEncoding()**  
**network.set\_callbackEncoding()**

**YNetwork**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
function set_callbackEncoding( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKENCODING\_FORM, Y\_CALLBACKENCODING\_JSON, Y\_CALLBACKENCODING\_JSON\_ARRAY, Y\_CALLBACKENCODING\_CSV et Y\_CALLBACKENCODING\_YOCTO\_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMaxDelay()**  
**network→setCallbackMaxDelay()**  
**network.set\_callbackMaxDelay()**

**YNetwork**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

```
function set_callbackMaxDelay( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant l'attente maximale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMethod()**  
**network→setCallbackMethod()**  
**network.set\_callbackMethod()**

**YNetwork**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
function set_callbackMethod( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** une valeur parmi Y\_CALLBACKMETHOD\_POST, Y\_CALLBACKMETHOD\_GET et Y\_CALLBACKMETHOD\_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackMinDelay()**  
**network→setCallbackMinDelay()**  
**network.set\_callbackMinDelay()**

**YNetwork**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

```
function set_callbackMinDelay( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant l'attente minimale entre deux notifications par callback, en secondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_callbackUrl()**

**YNetwork**

**network→setCallbackUrl()network.set\_callbackUrl()**

---

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
function set_callbackUrl( ByVal newval As String) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>network→set_discoverable()</b>	<b>YNetwork</b>
<b>network→setDiscoverable()</b>	
<b>network.set_discoverable()</b>	

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

function **set\_discoverable( ByVal newval As Integer) As Integer**

**Paramètres :**

**newval** soit Y\_DISCOVERABLE\_FALSE, soit Y\_DISCOVERABLE\_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_logicalName()**  
**network→setLogicalName()**  
**network.set\_logicalName()**

**YNetwork**

Modifie le nom logique de l'interface réseau.

**function set\_logicalName( ByVal newval As String) As Integer**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'interface réseau.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**network→set\_primaryDNS()** **YNetwork**  
**network→setPrimaryDNS()network.set\_primaryDNS()**

---

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
function set_primaryDNS( ByVal newval As String) As Integer
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**newval** une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_secondaryDNS()**  
**network→setSecondaryDNS()**  
**network.set\_secondaryDNS()**

**YNetwork**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
function set_secondaryDNS( ByVal newval As String) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**network→set\_userPassword()**  
**network→setUserPassword()**  
**network.set\_userPassword()**

**YNetwork**

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

```
function set_userPassword( ByVal newval As String) As Integer
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→set\_wwwWatchdogDelay()**  
**network→setWwwWatchdogDelay()**  
**network.set\_wwwWatchdogDelay()**

**YNetwork**

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

function **set\_wwwWatchdogDelay( ByVal newval As Integer) As Integer**

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW. La plus petite durée non-nulle utilisable est 90 secondes.

**Paramètres :**

**newval** un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useDHCP()network.useDHCP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

```
function useDHCP( ) As Integer
```

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode saveToFlash() et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**fallbackIpAddr**              adresse IP à utiliser si aucun serveur DHCP ne répond  
**fallbackSubnetMaskLen** longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.  
**fallbackRouter**              adresse de la passerelle à utiliser si aucun serveur DHCP ne répond

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**network→useStaticIP()network.useStaticIP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

```
function useStaticIP( ) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ipAddress**      adresse IP à utiliser par le module

**subnetMaskLen** longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.

**router**            adresse IP de la passerelle à utiliser ("default gateway")

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.29. contrôle d'OS

L'objet OsControl permet de contrôler le système d'exploitation sur lequel tourne un VirtualHub. OsControl n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activé au lancement du VirtualHub, avec l'option -o.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_oscontrol.js'></script>
nodejs var yoctolib = require('yoctolib');
var YOsControl = yoctolib.YOsControl;
require_once('yocto_oscontrol.php');
cpp #include "yocto_oscontrol.h"
m #import "yocto_oscontrol.h"
pas uses yocto_oscontrol;
vb yocto_oscontrol.vb
cs yocto_oscontrol.cs
java import com.yoctopuce.YoctoAPI.YOsControl;
py from yocto_oscontrol import *

```

### Fonction globales

#### **yFindOsControl(func)**

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

#### **yFirstOsControl()**

Commence l'énumération des contrôle d'OS accessibles par la librairie.

### Méthodes des objets YOsControl

#### **oscontrol→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format TYPE (NAME )=SERIAL . FUNCTIONID.

#### **oscontrol→get\_advertisedValue()**

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

#### **oscontrol→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

#### **oscontrol→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

#### **oscontrol→get\_friendlyName()**

Retourne un identifiant global du contrôle d'OS au format NOM\_MODULE . NOM\_FONCTION.

#### **oscontrol→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **oscontrol→get\_functionId()**

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

#### **oscontrol→get\_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'OS au format SERIAL . FUNCTIONID.

#### **oscontrol→get\_logicalName()**

Retourne le nom logique du contrôle d'OS.

#### **oscontrol→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **oscontrol→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**oscontrol->get\_shutdownCountdown()**

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

**oscontrol->get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**oscontrol->isOnline()**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**oscontrol->isOnline\_async(callback, context)**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

**oscontrol->load(msValidity)**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**oscontrol->load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

**oscontrol->nextOsControl()**

Continue l'énumération des contrôle d'OS commencée à l'aide de yFirstOsControl( ).

**oscontrol->registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**oscontrol->set\_logicalName(newval)**

Modifie le nom logique du contrôle d'OS.

**oscontrol->set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**oscontrol->shutdown(secBeforeShutDown)**

Agende un arrêt de l'OS dans un nombre donné de secondes.

**oscontrol->wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YOsControl.FindOsControl() yFindOsControl()yFindOsControl()

YOsControl

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

```
function yFindOsControl( ByVal func As String) As YOsControl
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'OS soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YOsControl.isOnline()` pour tester si le contrôle d'OS est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

### Retourne :

un objet de classe `YOsControl` qui permet ensuite de contrôler le contrôle d'OS.

## YOsControl.FirstOsControl() yFirstOsControl()yFirstOsControl()

**YOsControl**

Commence l'énumération des contrôle d'OS accessibles par la librairie.

function **yFirstOsControl( ) As YOsControl**

Utiliser la fonction `YOsControl.nextOsControl( )` pour itérer sur les autres contrôle d'OS.

**Retourne :**

un pointeur sur un objet `YOsControl`, correspondant au premier contrôle d'OS accessible en ligne, ou null si il n'y a pas de contrôle d'OS disponibles.

**oscontrol→describe()oscontrol.describe()****YOsControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe( ) As String
```

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).

```
function get_advertisedValue( ) As String
```

**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.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

**oscontrol→get\_errorType()****YOsControl****oscontrol→errorType()oscontrol.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**oscontrol→functionDescriptor()**  
**oscontrol.get\_functionDescriptor()**

---

**YOsControl**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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

**oscontrol→get\_functionId()****YOsControl****oscontrol→functionId()oscontrol.get\_functionId()**

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

```
function get_functionId( ) As String
```

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()**  
**oscontrol→hardwareId()oscontrol.get\_hardwareId()**

---

**YOsControl**

Retourne l'identifiant matériel unique du contrôle d'OS au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'OS (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le contrôle d'OS (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**oscontrol→get\_logicalName()  
oscontrol→logicalName()  
oscontrol.get\_logicalName()****YOsControl**

Retourne le nom logique du contrôle d'OS.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du contrôle d'OS.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**oscontrol→get\_module()**

**YOsControl**

**oscontrol→module()oscontrol.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**oscontrol→get\_shutdownCountdown()**  
**oscontrol→shutdownCountdown()**  
**oscontrol.get\_shutdownCountdown()****YOsControl**

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

function **get\_shutdownCountdown( ) As Integer**

**Retourne :**

un entier représentant le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé

En cas d'erreur, déclenche une exception ou retourne **Y\_SHUTDOWNCOUNTDOWN\_INVALID**.

**oscontrol→get(userData)**

**YOsControl**

**oscontrol→userData()oscontrol.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**oscontrol→isOnline()****YOsControl**

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le contrôle d'OS est joignable, false sinon

**oscontrol→load()oscontrol.load()****YOsControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**oscontrol→nextOsControl()****YOsControl****oscontrol.nextOsControl()**

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

```
function nextOsControl( ) As YOsControl
```

**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.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**oscontrol→set\_logicalName()  
oscontrol→setLogicalName()  
oscontrol.set\_logicalName()****YOscControl**

Modifie le nom logique du contrôle d'OS.

```
function set_logicalName( ByVal newval As String) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**oscontrol→shutdown()oscontrol.shutdown()****YOsControl**

Agende un arrêt de l'OS dans un nombre donné de secondes.

```
function shutdown( ) As Integer
```

**Paramètres :**

**secBeforeShutDown** nombre de secondes avant l'arrêt

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.30. 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), en Watt, sous forme de nombre à virgule.

#### **power→get\_currentValue()**

Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.

#### **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 depuis le démarrage du module.

**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 depuis le démarrage du module.

**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.

#### **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.

#### **power→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **power→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **power→set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

#### **power→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPower.FindPower() yFindPower()yFindPower()

YPower

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

```
function yFindPower( ByVal func As String) As YPower
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de puissance électrique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance électrique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le capteur de puissance électrique sans ambiguïté

### Retourne :

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance électrique.

## **YPower.FirstPower() yFirstPower()yFirstPower()**

**YPower**

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

```
function yFirstPower( ) As YPower
```

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance électrique.

**Retourne :**

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance électrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance électrique disponibles.

**power→calibrateFromPoints()**  
**power.calibrateFromPoints()****YPower**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→describe()power.describe()****YPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format TYPE (NAME )=SERIAL .FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de puissance électrique (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**power→get\_advertisedValue()**  
**power→advertisedValue()**  
**power.get\_advertisedValue()**

**YPower**

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**power→get\_cosPhi()**  
**power→cosPhi()power.get\_cosPhi()****YPower**

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

```
function get_cosPhi( ) As Double
```

**Retourne :**

une valeur numérique représentant le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA)

En cas d'erreur, déclenche une exception ou retourne Y\_COSPHI\_INVALID.

**power→get\_currentRawValue()**  
**power→currentRawValue()**  
**power.get\_currentRawValue()**

**YPower**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

**power→get\_currentValue()**

**YPower**

**power→currentValue()power.get\_currentValue()**

Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**power→getErrorMessage()****YPower****power→errorMessage()power.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
function getErrorMessage( ) As String
```

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.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**power→functionDescriptor()**  
**power.get\_functionDescriptor()****YPower**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**power→get\_functionId()**

**YPower**

**power→functionId()power.get\_functionId()**

---

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

function **get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**power→get\_hardwareId()****YPower****power→hardwareId()power.get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de puissance électrique (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de puissance électrique (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**power→get\_highestValue()**

**YPower**

**power→highestValue()power.get\_highestValue()**

Retourne la valeur maximale observée pour la puissance électrique depuis le démarrage du module.

**function get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la puissance électrique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**power→get\_logFrequency()****YPower****power→logFrequency()power.get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( ) As String
```

**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.

```
function get_logicalName( ) As String
```

**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`.

---

<b>power→get_lowestValue()</b>	<b>YPower</b>
<b>power→lowestValue()power.get_lowestValue()</b>	

---

Retourne la valeur minimale observée pour la puissance électrique depuis le démarrage du module.

```
function get_lowestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la puissance électrique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**power→get\_meter()  
power→meter()power.get\_meter()****YPower**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

**function get\_meter( ) As Double**

Ce compteur est réinitialisé à chaque démarrage du module.

**Retourne :**

une valeur numérique représentant la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée

En cas d'erreur, déclenche une exception ou retourne Y\_METER\_INVALID.

**power→get\_meterTimer()****YPower****power→meterTimer()power.get\_meterTimer()**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

```
function get_meterTimer( ) As Integer
```

**Retourne :**

un entier représentant le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

En cas d'erreur, déclenche une exception ou retourne Y\_METERTIMER\_INVALID.

**power→get\_module()**  
**power→module()power.get\_module()**

---

YPower

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

## power→get\_recordedData() YPower

## power→recordedData() power.get\_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( ) As YDataSet
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

### Paramètres :

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

### Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**power→get\_reportFrequency()**  
**power→reportFrequency()**  
**power.get\_reportFrequency()**

**YPower**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**power→get\_resolution()**  
**power→resolution()power.get\_resolution()****YPower**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**power→get\_unit()**  
**power→unit()power.get\_unit()**

---

YPower

Retourne l'unité dans laquelle la puissance électrique est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la puissance électrique est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**power→get(userData)****YPower****power→userData() power.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**power→isOnline()power.isOnline()****YPower**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du capteur de puissance électrique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de puissance électrique est joignable, false sinon

**power→load()power.load()****YPower**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

**procedure loadCalibrationPoints( )****Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**power→nextPower()power.nextPower()****YPower**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

```
function nextPower( ) As YPower
```

**Retourne :**

un pointeur sur un objet `YPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**power→registerTimedReportCallback()  
power.registerTimedReportCallback()****YPower**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**power→registerValueCallback()**  
**power.registerValueCallback()****YPower**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

## power→reset()power.reset()

YPower

Réinitialise le compteur d'énergie.

```
function reset( ) As Integer
```

**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>power→set_highestValue()</b>	<b>YPower</b>
<b>power→setHighestValue()power.set_highestValue()</b>	

---

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**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.

**power→set\_logFrequency()** YPower  
**power→setLogFrequency()power.set\_logFrequency()**

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**power→set\_logicalName()** YPower  
**power→setLogicalName()power.set\_logicalName()**

---

Modifie le nom logique du capteur de puissance électrique.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**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.

**power→set\_reportFrequency()  
power→setReportFrequency()  
power.set\_reportFrequency()****YPower**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>power→set_resolution()</b>	<b>YPower</b>
<b>power→setResolution()power.set_resolution()</b>	

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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.

**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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.31. Interface de la fonction Pressure

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_pressure.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPressure = yoctolib.YPressure;
php require_once('yocto_pressure.php');
cpp #include "yocto_pressure.h"
m #import "yocto_pressure.h"
pas uses yocto_pressure;
vb yocto_pressure.vb
cs yocto_pressure.cs
java import com.yoctopuce.YoctoAPI.YPressure;
py from yocto_pressure import *

```

### Fonction globales

#### yFindPressure(func)

Permet de retrouver un capteur de pression d'après un identifiant donné.

#### yFirstPressure()

Commence l'énumération des capteurs de pression accessibles par la librairie.

### Méthodes des objets YPressure

#### pressure→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### pressure→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### pressure→get\_advertisedValue()

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

#### pressure→get\_currentRawValue()

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule.

#### pressure→get\_currentValue()

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

#### pressure→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### pressure→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

#### pressure→get\_friendlyName()

Retourne un identifiant global du capteur de pression au format NOM\_MODULE . NOM\_FONCTION.

#### pressure→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### pressure→get\_functionId()

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

#### pressure→get\_hardwareId()

Retourne l'identifiant matériel unique du capteur de pression au format SERIAL.FUNCTIONID.
<b>pressure→get_highestValue()</b>
Retourne la valeur maximale observée pour la pression depuis le démarrage du module.
<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 depuis le démarrage du module.
<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.
<b>pressure→set_logFrequency(newval)</b>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**pressure→set\_logicalName(newval)**

Modifie le nom logique du capteur de pression.

**pressure→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**pressure→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**pressure→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**pressure→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**pressure→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPressure.FindPressure()****YPressure****yFindPressure()yFindPressure()**

Permet de retrouver un capteur de pression d'après un identifiant donné.

```
function yFindPressure( ByVal func As String) As YPressure
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnLine()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence le capteur de pression sans ambiguïté

**Retourne :**

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

## YPressure.FirstPressure() yFirstPressure()yFirstPressure()

**YPressure**

Commence l'énumération des capteurs de pression accessibles par la librairie.

```
function yFirstPressure( ) As YPressure
```

Utiliser la fonction `YPressure.nextPressure( )` pour itérer sur les autres capteurs de pression.

**Retourne :**

un pointeur sur un objet `YPressure`, correspondant au premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

**pressure→calibrateFromPoints()**  
**pressure.calibrateFromPoints()****YPressure**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→describe()pressure.describe()****YPressure**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de pression (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pressure→get\_advertisedValue()**  
**pressure→advertisedValue()**  
**pressure.get\_advertisedValue()**

**YPressure**

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

function **get\_advertisedValue( ) As String**

**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.

---

<b>pressure→get_currentRawValue()</b>	<b>YPressure</b>
<b>pressure→currentRawValue()</b>	
<b>pressure.get_currentRawValue()</b>	

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

**pressure→get\_currentValue()****YPressure****pressure→currentValue()pressure.get\_currentValue()**

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTVALUE\_INVALID**.

**pressure→get\_errorMessage()**  
**pressure→errorMessage()**  
**pressure.get\_errorMessage()**

**YPressure**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

**pressure→get\_errorType()****YPressure****pressure→errorType()pressure.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**pressure→functionDescriptor()**  
**pressure.get\_functionDescriptor()**

---

**YPressure**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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

**pressure→get\_functionId()****YPressure****pressure→functionId()pressure.get\_functionId()**

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de pression (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pressure→get.hardwareId()**

**YPressure**

**pressure→hardwareId()pressure.get.hardwareId()**

---

Retourne l'identifiant matériel unique du capteur de pression au format SERIAL.FUNCTIONID.

**function get.hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de pression (par exemple RELAYLO1-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de pression (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**pressure→get\_highestValue()**  
**pressure→highestValue()**  
**pressure.get\_highestValue()**

**YPressure**

Retourne la valeur maximale observée pour la pression depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la pression depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**pressure→get\_logFrequency()**  
**pressure→logFrequency()**  
**pressure.get\_logFrequency()**

**YPressure**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**pressure→get\_logicalName()**

**YPressure**

**pressure→logicalName()pressure.get\_logicalName()**

Retourne le nom logique du capteur de pression.

```
function get_logicalName( ) As String
```

**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 depuis le démarrage du module.

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la pression depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**pressure→get\_module()****YPressure****pressure→module()pressure.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**pressure→get\_recordedData()**  
**pressure→recordedData()**  
**pressure.get\_recordedData()**

**YPressure**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

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.

**pressure→get\_reportFrequency()**  
**pressure→reportFrequency()**  
**pressure.get\_reportFrequency()**

**YPressure**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**pressure→get\_resolution()**

**YPressure**

**pressure→resolution()pressure.get\_resolution()**

---

Retourne la résolution des valeurs mesurées.

**function get\_resolution( ) As Double**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**pressure→get\_unit()****YPressure****pressure→unit()pressure.get\_unit()**

Retourne l'unité dans laquelle la pression est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la pression est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**pressure→get(userData)**

**YPressure**

**pressure→userData()pressure.get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**function get(userData) As Object**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pressure→isOnline()pressure.isOnline()****YPressure**

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du capteur de pression sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si le capteur de pression est joignable, `false` sinon

**pressure→load()**pressure.load()******YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

```
procedure loadCalibrationPoints( )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **pressure→nextPressure()pressure.nextPressure()**

**YPressure**

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

**function nextPressure( ) As YPressure**

**Retourne :**

un pointeur sur un objet `YPressure` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pressure→registerTimedReportCallback()**  
**pressure.registerTimedReportCallback()****YPressure**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pressure→set\_highestValue()**  
**pressure→setHighestValue()**  
**pressure.set\_highestValue()**

YPressure

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_logFrequency()**  
**pressure→setLogFrequency()**  
**pressure.set\_logFrequency()**

**YPressure**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_logicalName()**  
**pressure→setLogicalName()**  
**pressure.set\_logicalName()**

**YPressure**

Modifie le nom logique du capteur de pression.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_reportFrequency()**  
**pressure→setReportFrequency()**  
**pressure.set\_reportFrequency()**

YPressure

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pressure→set\_resolution()** YPressure  
**pressure→setResolution()pressure.set\_resolution()**

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.32. Interface de la fonction PwmInput

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_pwminput.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPwmInput = yoctolib.YPwmInput;
php require_once('yocto_pwminput.php');
cpp #include "yocto_pwminput.h"
m #import "yocto_pwminput.h"
pas uses yocto_pwminput;
vb yocto_pwminput.vb
cs yocto_pwminput.cs
java import com.yoctopuce.YoctoAPI.YPwmInput;
py from yocto_pwminput import *

```

### Fonction globales

#### yFindPwmInput(func)

Permet de retrouver un capteur de tension d'après un identifiant donné.

#### yFirstPwmInput()

Commence l'énumération des capteurs de tension accessibles par la librairie.

### Méthodes des objets YPwmInput

#### pwminput→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.

#### pwminput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### pwminput→get\_advertisedValue()

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

#### pwminput→get\_currentRawValue()

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

#### pwminput→get\_currentValue()

Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.

#### pwminput→get\_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

#### pwminput→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### pwminput→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

#### pwminput→get\_frequency()

Retourne la fréquence du PWM en Hz.

#### pwminput→get\_friendlyName()

Retourne un identifiant global du capteur de tension au format NOM\_MODULE . NOM\_FONCTION.

#### pwminput→get\_functionDescriptor()

	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>pwminput→get_functionId()</b>	Retourne l'identifiant matériel du capteur de tension, sans référence au module.
<b>pwminput→get_hardwareId()</b>	Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.
<b>pwminput→get_highestValue()</b>	Retourne la valeur maximale observée pour la tension depuis le démarrage du module.
<b>pwminput→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>pwminput→get_logicalName()</b>	Retourne le nom logique du capteur de tension.
<b>pwminput→get_lowestValue()</b>	Retourne la valeur minimale observée pour la tension depuis le démarrage du module.
<b>pwminput→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pwminput→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pwminput→get_period()</b>	Retourne la période du PWM en millisecondes.
<b>pwminput→get_pulseCounter()</b>	Retourne la valeur du compteur d'impulsions.
<b>pwminput→get_pulseDuration()</b>	Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.
<b>pwminput→get_pulseTimer()</b>	Retourne le timer du compteur d'impulsions (ms)
<b>pwminput→get_pwmReportMode()</b>	Retourne le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get_currentValue et les callback.
<b>pwminput→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>pwminput→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>pwminput→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>pwminput→get_unit()</b>	Retourne l'unité dans laquelle la valeur retornnée par get_currentValue et les callback est exprimée.
<b>pwminput→get_userData()</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>pwminput→isOnline()</b>	Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.
<b>pwminput→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.
<b>pwminput→load(msValidity)</b>	

### 3. Reference

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

#### **pwminput→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

#### **pwminput→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

#### **pwminput→nextPwmInput()**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstPwmInput()`.

#### **pwminput→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

#### **pwminput→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **pwminput→resetCounter()**

réinitialise le compteur d'impulsions et son timer

#### **pwminput→set\_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

#### **pwminput→set\_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

#### **pwminput→set\_logicalName(newval)**

Modifie le nom logique du capteur de tension.

#### **pwminput→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

#### **pwminput→set\_pwmReportMode(newval)**

Change le type de paramètre (fréquence, duty cycle, longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction `get_currentValue` et les callback.

#### **pwminput→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **pwminput→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **pwminput→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)`.

#### **pwminput→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.

## YPwmInput.FindPwmInput() yFindPwmInput()yFindPwmInput()

YPwmInput

Permet de retrouver un capteur de tension d'après un identifiant donné.

```
function yFindPwmInput( ByVal func As String) As YPwmInput
```

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 `YPwmInput.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 `YPwmInput` qui permet ensuite de contrôler le capteur de tension.

## YPwmInput.FirstPwmInput() yFirstPwmInput()yFirstPwmInput()

YPwmInput

Commence l'énumération des capteurs de tension accessibles par la librairie.

```
function yFirstPwmInput( ) As YPwmInput
```

Utiliser la fonction YPwmInput .nextPwmInput ( ) pour itérer sur les autres capteurs de tension.

**Retourne :**

un pointeur sur un objet YPwmInput, correspondant au premier capteur de tension accessible en ligne, ou null si il n'y a pas de capteurs de tension disponibles.

**pwminput→calibrateFromPoints()**  
**pwminput.calibrateFromPoints()****YPwmInput**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

procedure **calibrateFromPoints( )**

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.

**pwminput→describe()pwminput.describe()****YPwmInput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le capteur de tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**pwminput→get\_advertisedValue()**  
**pwminput→advertisedValue()**  
**pwminput.get\_advertisedValue()**

YPwmInput

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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.

**pwminput→get\_currentRawValue()**  
**pwminput→currentRawValue()**  
**pwminput.get\_currentRawValue()**

**YPwmInput**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**pwminput→get\_currentValue()**  
**pwminput→currentValue()**  
**pwminput.get\_currentValue()**

**YPwmInput**

Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.

**function get\_currentValue( ) As Double**

En fonction du réglage pwmReportMode, cela peut être soit la fréquence en Hz, le duty cycle en % ou encore la longueur d'impulsion en ms.

**Retourne :**

une valeur numérique représentant la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**pwminput→get\_dutyCycle()**

**YPwmInput**

**pwminput→dutyCycle()pwminput.get\_dutyCycle()**

---

Retourne le duty cycle du PWM, en pour cents.

```
function get_dutyCycle( ) As Double
```

**Retourne :**

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne **Y\_DUTYCYCLE\_INVALID**.

**pwminput→get\_errorMessage()**  
**pwminput→errorMessage()**  
**pwminput.get\_errorMessage()**

YPwmInput

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

**function get\_errorMessage( ) As String**

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.

---

**pwminput→get\_errorType()** **YPwmInput**  
**pwminput→errorType()pwminput.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function get_errorType( ) As YRETCODE
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

---

<b>pwminput→get_frequency()</b>	<b>YPwmInput</b>
<b>pwminput→frequency()pwminput.get_frequency()</b>	

---

Retourne la fréquence du PWM en Hz.

```
function get_frequency( ) As Double
```

**Retourne :**

une valeur numérique représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne Y\_FREQUENCY\_INVALID.

**pwminput→get\_functionDescriptor()**  
**pwminput→functionDescriptor()**  
**pwminput.get\_functionDescriptor()**

**YPwmInput**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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

**pwminput→get\_functionId()****YPwmInput****pwminput→functionId()pwminput.get\_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

```
function get_functionId( ) As String
```

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`.

**pwminput→get\_hardwareId()**

**YPwmInput**

**pwminput→hardwareId()pwminput.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

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.

**pwminput→get\_highestValue()**  
**pwminput→highestValue()**  
**pwminput.get\_highestValue()****YPwmInput**

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

```
function get_highestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**pwminput→get\_logFrequency()**  
**pwminput→logFrequency()**  
**pwminput.get\_logFrequency()**

**YPwmInput**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function get\_logFrequency( ) As String**

**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.

**pwminput→get\_logicalName()**  
**pwminput→logicalName()**  
**pwminput.get\_logicalName()**

YPwmInput

Retourne le nom logique du capteur de tension.

```
function get_logicalName( ) As String
```

**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.

**pwminput→get\_lowestValue()**  
**pwminput→lowestValue()**  
**pwminput.get\_lowestValue()**

**YPwmInput**

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**pwminput→get\_module()****YPwmInput****pwminput→module()pwminput.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` rentrée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**pwminput→get\_period()**

**YPwmInput**

**pwminput→period()pwminput.get\_period()**

---

Retourne la période du PWM en millisecondes.

**function get\_period( ) As Double**

**Retourne :**

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PERIOD\_INVALID.

**pwminput→get\_pulseCounter()**  
**pwminput→pulseCounter()**  
**pwminput.get\_pulseCounter()**

**YPwmInput**

Retourne la valeur du compteur d'impulsions.

**function get\_pulseCounter( ) As Long**

Ce compteur est en réalité incrémenté deux fois par période. Ce compteur est limité à 1 milliard.

**Retourne :**

un entier représentant la valeur du compteur d'impulsions

En cas d'erreur, déclenche une exception ou retourne Y\_PULSECOUNTERR\_INVALID.

**pwminput→get\_pulseDuration()**  
**pwminput→pulseDuration()**  
**pwminput.get\_pulseDuration()**

**YPwmInput**

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

**function get\_pulseDuration( ) As Double**

**Retourne :**

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_PULSEDURATION\_INVALID.

`pwminput→get_pulseTimer()`

`YPwmInput`

`pwminput→pulseTimer()pwminput.get_pulseTimer()`

Retourne le timer du compteur d'impulsions (ms)

```
function get_pulseTimer( ) As Long
```

**Retourne :**

un entier représentant le timer du compteur d'impulsions (ms)

En cas d'erreur, déclenche une exception ou retourne `Y_PULSE_TIMER_INVALID`.

**pwminput→get\_pwmReportMode()**  
**pwminput→pwmReportMode()**  
**pwminput.get\_pwmReportMode()**

**YPwmInput**

Retourne le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get\_currentValue et les callback.

function **get\_pwmReportMode( ) As Integer**

**Retourne :**

une valeur parmi Y\_PWMREPORTMODE\_PWM\_DUTYCYCLE, Y\_PWMREPORTMODE\_PWM\_FREQUENCY, Y\_PWMREPORTMODE\_PWM\_PULSEDURATION et Y\_PWMREPORTMODE\_PWM\_EDGECOUNT représentant le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get\_currentValue et les callback

En cas d'erreur, déclenche une exception ou retourne Y\_PWMREPORTMODE\_INVALID.

**pwminput→get\_recordedData()**  
**pwminput→recordedData()**  
**pwminput.get\_recordedData()**

YPwmInput

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

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.

**pwminput→get\_reportFrequency()**  
**pwminput→reportFrequency()**  
**pwminput.get\_reportFrequency()**

**YPwmInput**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

---

<b>pwminput→get_resolution()</b>	<b>YPwmInput</b>
<b>pwminput→resolution()pwminput.get_resolution()</b>	

---

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

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.

**pwminput→get\_unit()**  
**pwminput→unit()pwminput.get\_unit()**

**YPwmInput**

Retourne l'unité dans laquelle la valeur renvoyée par get\_currentValue et les callback est exprimée.

**function get\_unit( ) As String**

Cette unité dépend du réglage pwmReportMode.

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la valeur renvoyée par get\_currentValue et les callback est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**pwminput→get(userData)****YPwmInput****pwminput→userData()pwminput.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

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.

**pwminput→isOnline()pwminput.isOnline()****YPwmInput**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

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

**pwminput→load()pwminput.load()****YPwmInput**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

**pwminput→loadCalibrationPoints()  
pwminput.loadCalibrationPoints()****YPwmInput**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

procedure **loadCalibrationPoints( )**

**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.

**pwminput→nextPwmInput()****YPwmInput**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstPwmInput( )`.

```
function nextPwmInput( ) As YPwmInput
```

**Retourne :**

un pointeur sur un objet `YPwmInput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pwminput→registerTimedReportCallback()  
pwminput.registerTimedReportCallback()****YPwmInput**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

**pwminput→registerValueCallback()  
pwminput.registerValueCallback()****YPwmInput**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

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.

## **pwminput→resetCounter()pwminput.resetCounter()**

**YPwmInput**

réinitialise le compteur d'impulsions et son timer

```
function resetCounter( ) As Integer
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwminput→set\_highestValue()**  
**pwminput→setHighestValue()**  
**pwminput.set\_highestValue()**

YPwmInput

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**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.

**pwminput→set\_logFrequency()**  
**pwminput→setLogFrequency()**  
**pwminput.set\_logFrequency()**

**YPwmInput**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( ByVal newval As String) As Integer**

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.

**pwminput→set\_logicalName()**  
**pwminput→setLogicalName()**  
**pwminput.set\_logicalName()**

**YPwmInput**

Modifie le nom logique du capteur de tension.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

**pwminput→set\_lowestValue()**  
**pwminput→setLowestValue()**  
**pwminput.set\_lowestValue()**

YPwmInput

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**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.

**pwminput→set\_pwmReportMode()**  
**pwminput→setPwmReportMode()**  
**pwminput.set\_pwmReportMode()**

**YPwmInput**

Change le type de paramètre (fréquence, duty cycle, longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get\_currentValue et les callback.

function **set\_pwmReportMode( ByVal newval As Integer)** As Integer

Seule les six digit de droite du nombre de changement d'état sont transmis, pour les valeurs plus grandes que un million, utiliser get\_pulseCounter().

**Paramètres :**

**newval** une valeur parmi Y\_PWMREPORTMODE\_PWM\_DUTYCYCLE,  
Y\_PWMREPORTMODE\_PWM\_FREQUENCY,  
Y\_PWMREPORTMODE\_PWM\_PULSEDURATION et  
Y\_PWMREPORTMODE\_PWM\_EDGECOUNT

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwminput→set\_reportFrequency()**  
**pwminput→setReportFrequency()**  
**pwminput.set\_reportFrequency()**

YPwmInput

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

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.

**pwminput→set\_resolution()  
pwminput→setResolution()  
pwminput.set\_resolution()****YPwmInput**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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.

---

**pwminput→set(userData)** YPwmInput  
**pwminput→setUserData()** pwminput.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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

### 3.33. Interface de la fonction Pwm

La librairie de programmation Yoctopuce permet simplement de configurer, démarrer et arrêter le PWM.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pwmoutput.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YPwmOutput = yoctolib.YPwmOutput;
php	require_once('yocto_pwmoutput.php');
cpp	#include "yocto_pwmoutput.h"
m	#import "yocto_pwmoutput.h"
pas	uses yocto_pwmoutput;
vb	yocto_pwmoutput.vb
cs	yocto_pwmoutput.cs
java	import com.yoctopuce.YoctoAPI.YPwmOutput;
py	from yocto_pwmoutput import *

#### Fonction globales

##### yFindPwmOutput(func)

Permet de retrouver un PWM d'après un identifiant donné.

##### yFirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

#### Méthodes des objets YPwmOutput

##### pwmoutput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format TYPE ( NAME )=SERIAL.FUNCTIONID.

##### pwmoutput→dutyCycleMove(target, ms\_duration)

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

##### pwmoutput→get\_advertisedValue()

Retourne la valeur courante du PWM (pas plus de 6 caractères).

##### pwmoutput→get\_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

##### pwmoutput→get\_dutyCycleAtPowerOn()

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

##### pwmoutput→get\_enabled()

Retourne l'état de fonctionnement du PWM.

##### pwmoutput→get\_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

##### pwmoutput→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

##### pwmoutput→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

##### pwmoutput→get\_frequency()

Retourne la fréquence du PWM en Hz.

##### pwmoutput→get\_friendlyName()

Retourne un identifiant global du PWM au format NOM\_MODULE . NOM\_FONCTION.

##### pwmoutput→get\_functionDescriptor()

### 3. Reference

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>pwmoutput→get_functionId()</b> Retourne l'identifiant matériel du PWM, sans référence au module.
<b>pwmoutput→get_hardwareId()</b> Retourne l'identifiant matériel unique du PWM au format SERIAL.FUNCTIONID.
<b>pwmoutput→get_logicalName()</b> Retourne le nom logique du PWM.
<b>pwmoutput→get_module()</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pwmoutput→get_module_async(callback, context)</b> Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>pwmoutput→get_period()</b> Retourne la période du PWM en millisecondes.
<b>pwmoutput→get_pulseDuration()</b> Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.
<b>pwmoutput→get_userData()</b> Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>pwmoutput→isOnline()</b> Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.
<b>pwmoutput→isOnline_async(callback, context)</b> Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.
<b>pwmoutput→load(msValidity)</b> Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.
<b>pwmoutput→load_async(msValidity, callback, context)</b> Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.
<b>pwmoutput→nextPwmOutput()</b> Continue l'énumération des PWM commencée à l'aide de yFirstPwmOutput().
<b>pwmoutput→pulseDurationMove(ms_target, ms_duration)</b> Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.
<b>pwmoutput→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>pwmoutput→set_dutyCycle(newval)</b> Modifie le duty cycle du PWM, en pour cents.
<b>pwmoutput→set_dutyCycleAtPowerOn(newval)</b> Modifie le duty cycle du PWM au démarrage du module.
<b>pwmoutput→set_enabled(newval)</b> Démarrer ou arrête le PWM.
<b>pwmoutput→set_enabledAtPowerOn(newval)</b> Modifie l'état du fonctionnement du PWM à la mise sous tension du module.
<b>pwmoutput→set_frequency(newval)</b> Modifie la fréquence du PWM.
<b>pwmoutput→set_logicalName(newval)</b> Modifie le nom logique du PWM.
<b>pwmoutput→set_period(newval)</b> Modifie la période du PWM en millisecondes.

**pwmoutput→set\_pulseDuration(newval)**

Modifie la longueur des impulsions du PWM, en millisecondes.

**pwmoutput→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**pwmoutput→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPwmOutput.FindPwmOutput() yFindPwmOutput()yFindPwmOutput()

**YPwmOutput**

Permet de retrouver un PWM d'après un identifiant donné.

```
function yFindPwmOutput( ByVal func As String) As YPwmOutput
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le PWM soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmOutput.isOnLine()` pour tester si le PWM est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le PWM sans ambiguïté

### Retourne :

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

## YPwmOutput.FirstPwmOutput() yFirstPwmOutput()yFirstPwmOutput()

YPwmOutput

Commence l'énumération des PWM accessibles par la librairie.

```
function yFirstPwmOutput( ) As YPwmOutput
```

Utiliser la fonction YPwmOutput .nextPwmOutput( ) pour itérer sur les autres PWM.

**Retourne :**

un pointeur sur un objet YPwmOutput, correspondant au premier PWM accessible en ligne, ou null si il n'y a pas de PWM disponibles.

**pwmoutput→describe()pwmoutput.describe()****YPwmOutput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

```
une chaîne de caractères décrivant le PWM (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**pwmoutput→dutyCycleMove()**  
**pwmoutput.dutyCycleMove()****YPwmOutput**

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

```
function dutyCycleMove( ) As Integer
```

**Paramètres :**

**target** nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)

**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→get\_advertisedValue()**  
**pwmoutput→advertisedValue()**  
**pwmoutput.get\_advertisedValue()**

**YPwmOutput**

---

Retourne la valeur courante du PWM (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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\_ADVISEDVALUE\_INVALID.

**pwmoutput→get\_dutyCycle()****YPwmOutput****pwmoutput→dutyCycle()pwmoutput.get\_dutyCycle()**

Retourne le duty cycle du PWM, en pour cents.

```
function get_dutyCycle( ) As Double
```

**Retourne :**

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne Y\_DUTYCYCLE\_INVALID.

**pwmoutput→get\_dutyCycleAtPowerOn()**  
**pwmoutput→dutyCycleAtPowerOn()**  
**pwmoutput.get\_dutyCycleAtPowerOn()**

**YPwmOutput**

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

function **get\_dutyCycleAtPowerOn( ) As Double**

**Retourne :**

une valeur numérique représentant le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

En cas d'erreur, déclenche une exception ou retourne **Y\_DUTYCYLEATPOWERON\_INVALID**.

**pwmoutput→get\_enabled()**

**YPwmOutput**

**pwmoutput→enabled()pwmoutput.get\_enabled()**

Retourne l'état de fonctionnement du PWM.

function **get\_enabled( ) As Integer**

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du PWM

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**pwmoutput→get\_enabledAtPowerOn()**  
**pwmoutput→enabledAtPowerOn()**  
**pwmoutput.get\_enabledAtPowerOn()**

**YPwmOutput**

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

```
function get_enabledAtPowerOn( ) As Integer
```

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état de fonctionnement du PWM à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

**pwmoutput→get\_errorMessage()**  
**pwmoutput→errorMessage()**  
**pwmoutput.get\_errorMessage()**

**YPwmOutput**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

**pwmoutput→get\_errorType()** YPwmOutput  
**pwmoutput→errorType()pwmoutput.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
function get_errorType( ) As YRETCODE
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

**pwmoutput→get\_frequency()**

**YPwmOutput**

**pwmoutput→frequency()pwmoutput.get\_frequency()**

Retourne la fréquence du PWM en Hz.

```
function get_frequency( ) As Double
```

**Retourne :**

une valeur numérique représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne **Y\_FREQUENCY\_INVALID**.

**pwmoutput→get\_functionDescriptor()**  
**pwmoutput→functionDescriptor()**  
**pwmoutput.get\_functionDescriptor()**

**YPwmOutput**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**pwmoutput→get\_functionId()****YPwmOutput****pwmoutput→functionId()pwmoutput.get\_functionId()**

Retourne l'identifiant matériel du PWM, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**pwmoutput→get\_hardwareId()**  
**pwmoutput→hardwareId()**  
**pwmoutput.get\_hardwareId()**

---

**YPwmOutput**

Retourne l'identifiant matériel unique du PWM au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du PWM (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le PWM (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**pwmoutput→get\_logicalName()**  
**pwmoutput→logicalName()**  
**pwmoutput.get\_logicalName()**

YPwmOutput

Retourne le nom logique du PWM.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du PWM.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pwmoutput→get\_module()**

**YPwmOutput**

**pwmoutput→module()pwmoutput.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**pwmoutput→get\_period()****YPwmOutput****pwmoutput→period()pwmoutput.get\_period()**

Retourne la période du PWM en millisecondes.

```
function get_period( ) As Double
```

**Retourne :**

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_PERIOD\_INVALID.

**pwmoutput→get\_pulseDuration()**  
**pwmoutput→pulseDuration()**  
**pwmoutput.get\_pulseDuration()**

**YPwmOutput**

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

**function get\_pulseDuration( ) As Double**

**Retourne :**

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_PULSE\_DURATION\_INVALID**.

**pwmoutput→get(userData)****YPwmOutput****pwmoutput→userData()pwmoutput.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pwmoutput→isOnline()pwmoutput.isOnline()****YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du PWM sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le PWM est joignable, false sinon

**pwmoutput→load()pwmoutput.load()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→nextPwmOutput()**  
**pwmoutput.nextPwmOutput()**

---

**YPwmOutput**

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

**function nextPwmOutput( ) As YPwmOutput**

**Retourne :**

un pointeur sur un objet `YPwmOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pwmoutput→pulseDurationMove()**  
**pwmoutput.pulseDurationMove()****YPwmOutput**

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

function **pulseDurationMove( ) As Integer**

N'importe quel changement de fréquence, duty cycle, période ou encore de longueur d'impulsion annulera tout processus de transition en cours.

**Paramètres :**

**ms\_target** nouvelle longueur des impulsions à la fin de la transition (nombre flottant, représentant la longueur en millisecondes)

**ms\_duration** durée totale de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→registerValueCallback()  
pwmoutput.registerValueCallback()****YPwmOutput**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pwmoutput→set\_dutyCycle()**  
**pwmoutput→setDutyCycle()**  
**pwmoutput.set\_dutyCycle()**

YPwmOutput

Modifie le duty cycle du PWM, en pour cents.

```
function set_dutyCycle( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM, en pour cents

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_dutyCycleAtPowerOn()**  
**pwmoutput→setDutyCycleAtPowerOn()**  
**pwmoutput.set\_dutyCycleAtPowerOn()**

**YPwmOutput**

Modifie le duty cycle du PWM au démarrage du module.

```
function set_dutyCycleAtPowerOn( ByVal newval As Double) As Integer
```

N'oubliez pas d'appeler la méthode saveToFlash( ) du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur numérique représentant le duty cycle du PWM au démarrage du module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_enabled()**

**YPwmOutput**

**pwmoutput→setEnabled()pwmoutput.set\_enabled()**

Démarre ou arrête le PWM.

```
function set_enabled( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_enabledAtPowerOn()**  
**pwmoutput→setEnabledAtPowerOn()**  
**pwmoutput.set\_enabledAtPowerOn()**

**YPwmOutput**

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

**function set\_enabledAtPowerOn( ByVal newval As Integer) As Integer**

N'oubliez pas d'appeler la méthode saveToFlash( ) du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état du fonctionnement du PWM à la mise sous tension du module

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_frequency()**  
**pwmoutput→setFrequency()**  
**pwmoutput.set\_frequency()**

**YPwmOutput**

Modifie la fréquence du PWM.

```
function set_frequency( ByVal newval As Double) As Integer
```

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

**Paramètres :**

**newval** une valeur numérique représentant la fréquence du PWM

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_logicalName()**  
**pwmoutput→setLogicalName()**  
**pwmoutput.set\_logicalName()**

**YPwmOutput**

Modifie le nom logique du PWM.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du PWM.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set\_period()****YPwmOutput****pwmoutput→setPeriod()pwmoutput.set\_period()**

Modifie la période du PWM en millisecondes.

```
function set_period( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la période du PWM en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>pwmoutput→set_pulseDuration()</b>	<b>YPwmOutput</b>
<b>pwmoutput→setPulseDuration()</b>	
<b>pwmoutput.set_pulseDuration()</b>	

Modifie la longueur des impulsions du PWM, en millisecondes.

```
function set_pulseDuration( ByVal newval As Double) As Integer
```

Attention, la longueur d'une impulsion ne peut pas être plus grande que la période, sinon la longueur sera automatiquement tronquée à la période.

**Paramètres :**

**newval** une valeur numérique représentant la longueur des impulsions du PWM, en millisecondes

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmoutput→set(userData)**  
**pwmoutput→setUserData()**  
**pwmoutput.set(userData)**

**YPwmOutput**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

procedure **set(userData)** ByVal **data** As Object

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.34. Interface de la fonction PwmPowerSource

La librairie de programmation Yoctopuce permet de configurer la source de tension utilisée par tous les PWM situés sur un même module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_pwmpowersource.js'></script>
nodejs var yoctolib = require('yoctolib');
var YPwmPowerSource = yoctolib.YPwmPowerSource;
php require_once('yocto_pwmpowersource.php');
cpp #include "yocto_pwmpowersource.h"
m #import "yocto_pwmpowersource.h"
pas uses yocto_pwmpowersource;
vb yocto_pwmpowersource.vb
cs yocto_pwmpowersource.cs
java import com.yoctopuce.YoctoAPI.YPwmPowerSource;
py from yocto_pwmpowersource import *

```

### Fonction globales

#### **yFindPwmPowerSource(func)**

Permet de retrouver une source de tension d'après un identifiant donné.

#### **yFirstPwmPowerSource()**

Commence l'énumération des Source de tension accessibles par la librairie.

### Méthodes des objets YPwmPowerSource

#### **pwmpowersource→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **pwmpowersource→get\_advertisedValue()**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

#### **pwmpowersource→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

#### **pwmpowersource→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

#### **pwmpowersource→get\_friendlyName()**

Retourne un identifiant global de la source de tension au format NOM\_MODULE . NOM\_FONCTION.

#### **pwmpowersource→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **pwmpowersource→get\_functionId()**

Retourne l'identifiant matériel de la source de tension, sans référence au module.

#### **pwmpowersource→get\_hardwareId()**

Retourne l'identifiant matériel unique de la source de tension au format SERIAL . FUNCTIONID.

#### **pwmpowersource→get\_logicalName()**

Retourne le nom logique de la source de tension.

#### **pwmpowersource→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **pwmpowersource→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**pwmpowersource→get\_powerMode()**

Retourne la source de tension utilisé par tous les PWM du même module.

**pwmpowersource→get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**pwmpowersource→isOnline()**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**pwmpowersource→isOnline\_async(callback, context)**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**pwmpowersource→load(msValidity)**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

**pwmpowersource→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

**pwmpowersource→nextPwmPowerSource()**

Continue l'énumération des Source de tension commencée à l'aide de yFirstPwmPowerSource( ).

**pwmpowersource→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**pwmpowersource→set\_logicalName(newval)**

Modifie le nom logique de la source de tension.

**pwmpowersource→set\_powerMode(newval)**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

**pwmpowersource→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**pwmpowersource→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YPwmPowerSource.FindPwmPowerSource() yFindPwmPowerSource()yFindPwmPowerSource()

YPwmPowerSource

Permet de retrouver une source de tension d'après un identifiant donné.

```
function yFindPwmPowerSource( ByVal func As String) As YPwmPowerSource
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmPowerSource.isOnLine()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence la source de tension sans ambiguïté

### Retourne :

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

**YPwmPowerSource.FirstPwmPowerSource()****yFirstPwmPowerSource()yFirstPwmPowerSource()****YPwmPowerSource**

Commence l'énumération des Source de tension accessibles par la librairie.

```
function yFirstPwmPowerSource( ) As YPwmPowerSource
```

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource()` pour itérer sur les autres Source de tension.

**Retourne :**

un pointeur sur un objet `YPwmPowerSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de Source de tension disponibles.

**pwmpowersource→describe()**  
**pwmpowersource.describe()****YPwmPowerSource**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format TYPE (NAME )=SERIAL.FUNCTIONID.

**function describe( ) As String**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant la source de tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**pwmpowersource→get\_advertisedValue()**  
**pwmpowersource→advertisedValue()**  
**pwmpowersource.get\_advertisedValue()**

**YPwmPowerSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**pwmpowersource→get\_errorMessage()**  
**pwmpowersource→errorMessage()**  
**pwmpowersource.get\_errorMessage()**

**YPwmPowerSource**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

**pwmpowersource→get\_errorType()**  
**pwmpowersource→errorType()**  
**pwmpowersource.get\_errorType()**

**YPwmPowerSource**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

**function get\_errorType( ) As YRETCODE**

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\_functionDescriptor()**  
**pwmpowersource→functionDescriptor()**  
**pwmpowersource.get\_functionDescriptor()**

---

**YPwmPowerSource**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**pwmpowersource→get\_functionId()**  
**pwmpowersource→functionId()**  
**pwmpowersource.get\_functionId()**

**YPwmPowerSource**

Retourne l'identifiant matériel de la source de tension, sans référence au module.

function **get\_functionId( ) As String**

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.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la source de tension (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la source de tension (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**pwmpowersource→get\_logicalName()**  
**pwmpowersource→logicalName()**  
**pwmpowersource.get\_logicalName()**

**YPwmPowerSource**

Retourne le nom logique de la source de tension.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de la source de tension.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**pwmpowersource→get\_module()**  
**pwmpowersource→module()**  
**pwmpowersource.get\_module()**

**YPwmPowerSource**

Retourne l'objet **YModule** correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de **YModule** retournée ne sera pas joignable.

**Retourne :**

une instance de **YModule**

**pwmpowersource→get\_powerMode()**  
**pwmpowersource→powerMode()**  
**pwmpowersource.get\_powerMode()**

**YPwmPowerSource**

Retourne la source de tension utilisé par tous les PWM du même module.

```
function get_powerMode( ) As Integer
```

**Retourne :**

une valeur parmi Y\_POWERMODE\_USB\_5V, Y\_POWERMODE\_USB\_3V, Y\_POWERMODE\_EXT\_V et  
Y\_POWERMODE\_OPNDRN représentant la source de tension utilisé par tous les PWM du même module

En cas d'erreur, déclenche une exception ou retourne Y\_POWERMODE\_INVALID.

**pwmpowersource→get(userData)**  
**pwmpowersource→userData()**  
**pwmpowersource.get(userData)**

**YPwmPowerSource**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**pwmpowersource→isOnline()**  
**pwmpowersource.isOnline()****YPwmPowerSource**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de la source de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si la source de tension est joignable, false sinon

**pwmpowersource→load()pwmpowersource.load()****YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→nextPwmPowerSource()**  
**pwmpowersource.nextPwmPowerSource()****YPwmPowerSource**

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

```
function nextPwmPowerSource( ) As YPwmPowerSource
```

**Retourne :**

un pointeur sur un objet `YPwmPowerSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**pwmpowersource→registerValueCallback()  
pwmpowersource.registerValueCallback()****YPwmPowerSource**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**pwmpowersource→set\_logicalName()**  
**pwmpowersource→setLogicalName()**  
**pwmpowersource.set\_logicalName()**

**YPwmPowerSource**

Modifie le nom logique de la source de tension.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de la source de tension.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**pwmpowersource→set\_powerMode()**  
**pwmpowersource→setPowerMode()**  
**pwmpowersource.set\_powerMode()**

**YPwmPowerSource**

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

function **set\_powerMode( ByVal newval As Integer) As Integer**

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).

procedure **set(userData)** ByVal **data** As Object

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.35. Interface du quaternion

La class YQt de la librairie Yoctopuce permet d'accéder à l'estimation de l'orientation tridimensionnelle du Yocto-3D sous forme d'un quaternion. Il n'est en général pas nécessaire d'y accéder directement, la classe YGyro offrant une abstraction de plus haut niveau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_gyro.js'></script>
nodejs var yoctolib = require('yoctolib');
var YGyro = yoctolib.YGyro;
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), en unités, sous forme de nombre à virgule.

#### qt→get\_currentValue()

Retourne la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule.

#### qt→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

#### qt→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

#### qt→get\_friendlyName()

Retourne un identifiant global de l'élément de quaternion au format NOM\_MODULE . NOM\_FONCTION.

#### qt→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### qt→get\_functionId()

	Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.
<b>qt→get_hardwareId()</b>	Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL.FUNCTIONID.
<b>qt→get_highestValue()</b>	Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.
<b>qt→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>qt→get_logicalName()</b>	Retourne le nom logique de l'élément de quaternion.
<b>qt→get_lowestValue()</b>	Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.
<b>qt→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>qt→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>qt→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>qt→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>qt→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>qt→get_unit()</b>	Retourne l'unité dans laquelle la coordonnée est exprimée.
<b>qt→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>qt→isOnline()</b>	Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.
<b>qt→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.
<b>qt→load(msValidity)</b>	Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.
<b>qt→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>qt→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.
<b>qt→nextQt()</b>	Continue l'énumération des éléments de quaternion commencée à l'aide de yFirstQt( ).
<b>qt→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>qt→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>qt→set_highestValue(newval)</b>	

### 3. Reference

---

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)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**qt→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YQt.FindQt()****YQt****yFindQt()yFindQt()**

Permet de retrouver un élément de quaternion d'après un identifiant donné.

```
function yFindQt( ByVal func As String) As YQt
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnLine()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

**Retourne :**

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

## YQt.FirstQt() yFirstQt()yFirstQt()

YQt

Commence l'énumération des éléments de quaternion accessibles par la librairie.

```
function yFirstQt( ) As YQt
```

Utiliser la fonction YQt .nextQt( ) pour itérer sur les autres éléments de quaternion.

**Retourne :**

un pointeur sur un objet YQt, correspondant au premier élément de quaternion accessible en ligne, ou null si il n'y a pas de éléments de quaternion disponibles.

**qt→calibrateFromPoints()qt.calibrateFromPoints()****YQt**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

procedure **calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→describe()qt.describe()****YQt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'élément de quaternion (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**qt→get\_advertisedValue()****YQt****qt→advertisedValue()qt.get\_advertisedValue()**

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**qt→get\_currentRawValue()** YQt  
**qt→currentRawValue()qt.get\_currentRawValue()**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule.

```
function get_currentRawValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule

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, en unités, sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule

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.

```
function get_errorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

**qt→get\_errorType()****YQt****qt→errorType()qt.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()** YQt  
**qt→functionDescriptor()qt.get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**qt→get\_functionId()****YQt****qt→functionId()qt.get\_functionId()**

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**qt→get\_hardwareId()****YQt****qt→hardwareId()qt.get\_hardwareId()**

Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'élément de quaternion (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'élément de quaternion (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**qt→get\_highestValue()****YQt****qt→highestValue()qt.get\_highestValue()**

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**qt→get\_logFrequency()  
qt→logFrequency()qt.get\_logFrequency()****YQt**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( ) As String
```

**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.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'élément de quaternion.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**qt→get\_lowestValue()**

**YQt**

**qt→lowestValue()qt.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

```
function get_lowestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**qt→get\_module()****YQt****qt→module()qt.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

<b>qt→get_recordedData()</b>	<b>YQt</b>
<b>qt→recordedData()qt.get_recordedData()</b>	

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**qt→get\_reportFrequency()****YQt****qt→reportFrequency()qt.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( ) As String
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**qt→get\_resolution()****YQt****qt→resolution()qt.get\_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**qt→get\_unit()****YQt****qt→unit()qt.get\_unit()**

Retourne l'unité dans laquelle la coordonnée est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la coordonnée est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**qt→get(userData)****YQt****qt→userData()qt.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**qt→isOnline()qt.isOnline()****YQt**

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'élément de quaternion est joignable, `false` sinon

**qt→load()qt.load()****YQt**

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

procedure **loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **qt→nextQt()qt.nextQt()**

**YQt**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

```
function nextQt( ) As YQt
```

**Retourne :**

un pointeur sur un objet `YQt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**qt→registerTimedReportCallback()  
qt.registerTimedReportCallback()****YQt**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback( ) As Integer**

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.

**qt→registerValueCallback()**  
**qt.registerValueCallback()****YQt**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

**qt→set\_highestValue()**  
**qt→setHighestValue()qt.set\_highestValue()****YQt**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**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.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_logicalName()****YQt****qt→setLogicalName()qt.set\_logicalName()**

Modifie le nom logique de l'élément de quaternion.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'élément de quaternion.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

### **qt→set\_lowestValue()**

YQt

### **qt→setLowestValue()qt.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set\_reportFrequency()**

YQt

**qt→setReportFrequency()qt.set\_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>qt→set_resolution()</b>	<b>YQt</b>
<b>qt→setResolution()qt.set_resolution()</b>	

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**qt→set(userData)****YQt****qt→setUserData()qt.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.36. Interface de la fonction Horloge Temps Réel

La fonction RealTimeClock fournit la date et l'heure courante de manière persistante, même en cas de coupure de courant de plusieurs jours. Elle est le fondement des fonctions de réveil automatique implémentées par le WakeUpScheduler. L'heure courante peut représenter aussi bien une heure locale qu'une heure UTC, mais aucune adaptation automatique n'est faite au changement d'heure été/hiver.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_realtimeclock.js'></script>
nodejs var yoctolib = require('yoctolib');
var YRealTimeClock = yoctolib.YRealTimeClock;
php require_once('yocto_realtimeclock.php');
cpp #include "yocto_realtimeclock.h"
m #import "yocto_realtimeclock.h"
pas uses yocto_realtimeclock;
vb yocto_realtimeclock.vb
cs yocto_realtimeclock.cs
java import com.yoctopuce.YoctoAPI.YRealTimeClock;
py from yocto_realtimeclock import *

```

### Fonction globales

#### yFindRealTimeClock(func)

Permet de retrouver une horloge d'après un identifiant donné.

#### yFirstRealTimeClock()

Commence l'énumération des horloges accessibles par la librairie.

### Méthodes des objets YRealTimeClock

#### realtimeclock→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### realtimeclock→get\_advertisedValue()

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

#### realtimeclock→get\_dateTime()

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

#### realtimeclock→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

#### realtimeclock→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

#### realtimeclock→get\_friendlyName()

Retourne un identifiant global de l'horloge au format NOM\_MODULE . NOM\_FONCTION.

#### realtimeclock→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### realtimeclock→get\_functionId()

Retourne l'identifiant matériel de l'horloge, sans référence au module.

#### realtimeclock→get\_hardwareId()

Retourne l'identifiant matériel unique de l'horloge au format SERIAL . FUNCTIONID.

#### realtimeclock→get\_logicalName()

Retourne le nom logique de l'horloge.

#### realtimeclock→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**realtimeclock→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**realtimeclock→get\_timeSet()**

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

**realtimeclock→get\_unixTime()**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

**realtimeclock→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**realtimeclock→get\_utcOffset()**

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**realtimeclock→isOnline()**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**realtimeclock→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**realtimeclock→load(msValidity)**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**realtimeclock→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

**realtimeclock→nextRealTimeClock()**

Continue l'énumération des horloge commencée à l'aide de yFirstRealTimeClock( ).

**realtimeclock→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**realtimeclock→set\_logicalName(newval)**

Modifie le nom logique de l'horloge.

**realtimeclock→set\_unixTime(newval)**

Modifie l'heure courante.

**realtimeclock→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**realtimeclock→set\_utcOffset(newval)**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

**realtimeclock→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YRealTimeClock.FindRealTimeClock() yFindRealTimeClock()yFindRealTimeClock()

**YRealTimeClock**

Permet de retrouver une horloge d'après un identifiant donné.

```
function yFindRealTimeClock( ByVal func As String) As YRealTimeClock
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnline()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence l'horloge sans ambiguïté

### Retourne :

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

**YRealTimeClock.FirstRealTimeClock()****yFirstRealTimeClock()yFirstRealTimeClock()****YRealTimeClock**

Commence l'énumération des horloge accessibles par la librairie.

```
function yFirstRealTimeClock( ) As YRealTimeClock
```

Utiliser la fonction `YRealTimeClock.nextRealTimeClock( )` pour itérer sur les autres horloge.

**Retourne :**

un pointeur sur un objet `YRealTimeClock`, correspondant à la première horloge accessible en ligne, ou null si il n'y a pas de horloge disponibles.

**realtimeclock→describe()realtimeclock.describe()****YRealTimeClock**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe() As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'horloge (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**realtimeclock→get\_advertisedValue()**  
**realtimeclock→advertisedValue()**  
**realtimeclock.get\_advertisedValue()**

**YRealTimeClock**

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**realtimeclock→getDateTime()**  
**realtimeclock→dateTime()**  
**realtimeclock.getDateTime()**

---

**YRealTimeClock**

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

```
function getDateTime( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

En cas d'erreur, déclenche une exception ou retourne Y\_DATETIME\_INVALID.

**realtimeclock→get\_errorMessage()**  
**realtimeclock→errorMessage()**  
**realtimeclock.get\_errorMessage()**

**YRealTimeClock**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

**realtimeclock→get\_errorType()**  
**realtimeclock→errorType()**  
**realtimeclock.get\_errorType()**

**YRealTimeClock**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

**function get\_errorType( ) As YRETCODE**

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\_functionDescriptor()**  
**realtimeclock→functionDescriptor()**  
**realtimeclock.get\_functionDescriptor()**

**YRealTimeClock**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**realtimeclock→get\_functionId()**  
**realtimeclock→functionId()**  
**realtimeclock.get\_functionId()**

---

**YRealTimeClock**

Retourne l'identifiant matériel de l'horloge, sans référence au module.

**function get\_functionId( ) As String**

Par example `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**realtimeclock→get\_hardwareId()**  
**realtimeclock→hardwareId()**  
**realtimeclock.get\_hardwareId()**

**YRealTimeClock**

Retourne l'identifiant matériel unique de l'horloge au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'horloge (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'horloge (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**realtimeclock→get\_logicalName()**  
**realtimeclock→logicalName()**  
**realtimeclock.get\_logicalName()**

---

**YRealTimeClock**

Retourne le nom logique de l'horloge.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'horloge.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**realtimeclock→get\_module()****YRealTimeClock****realtimeclock→module()realtimeclock.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**realtimeclock→get\_timeSet()**

**YRealTimeClock**

**realtimeclock→timeSet()realtimeclock.get\_timeSet()**

---

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

```
function get_timeSet( ) As Integer
```

**Retourne :**

soit Y\_TIMESET\_FALSE, soit Y\_TIMESET\_TRUE, selon vrai si l'horloge à été mise à l'heure, sinon faux

En cas d'erreur, déclenche une exception ou retourne Y\_TIMESET\_INVALID.

**realtimeclock→get\_unixTime()**  
**realtimeclock→unixTime()**  
**realtimeclock.get\_unixTime()**

**YRealTimeClock**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

function **get\_unixTime( ) As Long**

**Retourne :**

un entier représentant l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970)

En cas d'erreur, déclenche une exception ou retourne **Y\_UNIXTIME\_INVALID**.

**realtimeclock→get(userData)**  
**realtimeclock→userData()**  
**realtimeclock.get(userData)**

**YRealTimeClock**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**realtimeclock→get\_utcOffset()**  
**realtimeclock→utcOffset()**  
**realtimeclock.get\_utcOffset()**

**YRealTimeClock**

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

```
function get_utcOffset( ) As Integer
```

**Retourne :**

un entier représentant le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne Y\_UTCOFFSET\_INVALID.

**realtimeclock→isOnline()realtimeclock.isOnline()****YRealTimeClock**

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'horloge est joignable, false sinon

**realtimeclock→load()realtimeclock.load()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→nextRealTimeClock()**  
**realtimeclock.nextRealTimeClock()**

---

**YRealTimeClock**

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

```
function nextRealTimeClock( ) As YRealTimeClock
```

**Retourne :**

un pointeur sur un objet `YRealTimeClock` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**realtimeclock→registerValueCallback()**  
**realtimeclock.registerValueCallback()****YRealTimeClock**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**realtimeclock→set\_logicalName()**  
**realtimeclock→setLogicalName()**  
**realtimeclock.set\_logicalName()**

**YRealTimeClock**

Modifie le nom logique de l'horloge.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'horloge.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→set\_unixTime()**  
**realtimeclock→setUnixTime()**  
**realtimeclock.set\_unixTime()**

**YRealTimeClock**

Modifie l'heure courante.

```
function set_unixTime( ByVal newval As Long) As Integer
```

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).

**procedure set(userData( ByVal data As Object)**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**realtimeclock→set\_utcOffset()**  
**realtimeclock→setUtcOffset()**  
**realtimeclock.set\_utcOffset()**

**YRealTimeClock**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
function set_utcOffset( ByVal newval As Integer) As Integer
```

Le décallage est automatiquement arrondi au quart d'heure le plus proche. Si l'heure UTC est connue, l'heure courante sera automatiquement adaptée en fonction du décalage choisi.

**Paramètres :**

**newval** un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.37. Configuration du référentiel

Cette classe permet de configurer l'orientation dans laquelle le Yocto-3D est utilisé, afin que les fonctions d'orientation relatives au plan de la surface terrestre utilisent le référentiel approprié. La classe offre aussi un processus de recalibration tridimensionnel des capteurs, permettant de compenser les variations locales de l'accélération terrestre et d'améliorer la précision des capteurs d'inclinaisons.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_refframe.js'></script>
nodejs var yoctolib = require('yoctolib');
var YRefFrame = yoctolib.YRefFrame;
php require_once('yocto_refframe.php');
cpp #include "yocto_refframe.h"
m #import "yocto_refframe.h"
pas uses yocto_refframe;
vb yocto_refframe.vb
cs yocto_refframe.cs
java import com.yoctopuce.YoctoAPI.YRefFrame;
py from yocto_refframe import *

```

### Fonction globales

#### yFindRefFrame(func)

Permet de retrouver un référentiel d'après un identifiant donné.

#### yFirstRefFrame()

Commence l'énumération des référentiels accessibles par la librairie.

### Méthodes des objets YRefFrame

#### refframe→cancel3DCalibration()

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

#### refframe→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### refframe→get\_3DCalibrationHint()

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode start3DCalibration.

#### refframe→get\_3DCalibrationLogMsg()

Retourne le dernier message de log produit par le processus de calibration.

#### refframe→get\_3DCalibrationProgress()

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode start3DCalibration.

#### refframe→get\_3DCalibrationStage()

Retourne l'index de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

#### refframe→get\_3DCalibrationStageProgress()

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

#### refframe→get\_advertisedValue()

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

#### refframe→get\_bearing()

Retourne le cap de référence utilisé par le compas.

#### **refframe→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

#### **refframe→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

#### **refframe→get\_friendlyName()**

Retourne un identifiant global du référentiel au format NOM\_MODULE . NOM\_FONCTION.

#### **refframe→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **refframe→get\_functionId()**

Retourne l'identifiant matériel du référentiel, sans référence au module.

#### **refframe→get\_hardwareId()**

Retourne l'identifiant matériel unique du référentiel au format SERIAL . FUNCTIONID.

#### **refframe→get\_logicalName()**

Retourne le nom logique du référentiel.

#### **refframe→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **refframe→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **refframe→get\_mountOrientation()**

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

#### **refframe→get\_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

#### **refframe→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### **refframe→isOnline()**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

#### **refframe→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

#### **refframe→load(msValidity)**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

#### **refframe→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

#### **refframe→more3DCalibration()**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.

#### **refframe→nextRefFrame()**

Continue l'énumération des référentiels commencée à l'aide de yFirstRefFrame( ).

#### **refframe→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### **refframe→save3DCalibration()**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

#### **refframe→set\_bearing(newval)**

### 3. Reference

---

Modifie le cap de référence utilisé par le compas.

**refframe→set\_logicalName(newval)**

Modifie le nom logique du référentiel.

**refframe→set\_mountPosition(position, orientation)**

Modifie le référentiel de la boussole et des inclinomètres.

**refframe→set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**refframe→start3DCalibration()**

Initie le processus de calibration tridimensionnelle des capteurs.

**refframe→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YRefFrame.FindRefFrame() yFindRefFrame()yFindRefFrame()

YRefFrame

Permet de retrouver un référentiel d'après un identifiant donné.

```
function yFindRefFrame( ByVal func As String) As YRefFrame
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRefFrame.isOnline()` pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le référentiel sans ambiguïté

### Retourne :

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

## **YRefFrame.FirstRefFrame() yFirstRefFrame()yFirstRefFrame()**

---

**YRefFrame**

Commence l'énumération des référentiels accessibles par la librairie.

```
function yFirstRefFrame( ) As YRefFrame
```

Utiliser la fonction `YRefFrame.nextRefFrame( )` pour itérer sur les autres référentiels.

**Retourne :**

un pointeur sur un objet `YRefFrame`, correspondant au premier référentiel accessible en ligne, ou `null` si il n'y a pas de référentiels disponibles.

**refframe→cancel3DCalibration()**  
**refframe.cancel3DCalibration()****YRefFrame**

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

function **cancel3DCalibration( ) As Integer**

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→describe()refframe.describe()****YRefFrame**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**function describe( ) As String**

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.

function **get\_3DCalibrationHint( ) As String**

**Retourne :**  
une chaîne de caractères.

**refframe→get\_3DCalibrationLogMsg()**  
**refframe→3DCalibrationLogMsg()**  
**refframe.get\_3DCalibrationLogMsg()**

---

**YRefFrame**

Retourne le dernier message de log produit par le processus de calibration.

```
function get_3DCalibrationLogMsg( ) As String
```

Si aucun nouveau message n'est disponible, retourne une chaîne vide.

**Retourne :**  
une chaîne de caractères.

**refframe→get\_3DCalibrationProgress()**  
**refframe→3DCalibrationProgress()**  
**refframe.get\_3DCalibrationProgress()**

**YRefFrame**

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode start3DCalibration.

function **get\_3DCalibrationProgress( ) As Integer**

**Retourne :**

une nombre entier entre 0 (pas commencé) et 100 (terminé).

**refframe→get\_3DCalibrationStage()**  
**refframe→3DCalibrationStage()**  
**refframe.get\_3DCalibrationStage()**

**YRefFrame**

Retourne l'index de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

function **get\_3DCalibrationStage( ) As Integer**

**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.

function **get\_3DCalibrationStageProgress( ) As Integer**

**Retourne :**

une nombre entier entre 0 (pas commencé) et 100 (terminé).

**refframe→get\_advertisedValue()**  
**refframe→advertisedValue()**  
**refframe.get\_advertisedValue()**

**YRefFrame**

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

**function get\_advertisedValue( ) As String**

**Retourne :**

une chaîne de caractères représentant la valeur courante du référentiel (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**refframe→get\_bearing()****YRefFrame****refframe→bearing()refframe.get\_bearing()**

Retourne le cap de référence utilisé par le compas.

```
function get_bearing( ) As Double
```

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.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

**refframe→get\_errorType()****YRefFrame****refframe→errorType()refframe.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

```
function get_errorType( ) As YRETCODE
```

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_functionDescriptor()`  
`refframe→functionDescriptor()`  
`refframe.get_functionDescriptor()`

**YRefFrame**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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

**refframe→get\_functionId()****YRefFrame****refframe→functionId()refframe.get\_functionId()**

Retourne l'identifiant matériel du référentiel, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**refframe→get\_hardwareId()**

**YRefFrame**

**refframe→hardwareId()refframe.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du référentiel au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du référentiel (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le référentiel (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**refframe→get\_logicalName()**

**YRefFrame**

**refframe→logicalName()refframe.get\_logicalName()**

Retourne le nom logique du référentiel.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du référentiel.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

**refframe→get\_module()**

**YRefFrame**

**refframe→module()refframe.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**refframe→get\_mountOrientation()**  
**refframe→mountOrientation()**  
**refframe.get\_mountOrientation()**

**YRefFrame**

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

function **get\_mountOrientation( ) As Y\_MOUNTORIENTATION**

**Retourne :**

une valeur parmi l'énumération **Y\_MOUNTORIENTATION** (**Y\_MOUNTORIENTATION\_TWELVE**, **Y\_MOUNTORIENTATION\_THREE**, **Y\_MOUNTORIENTATION\_SIX**, **Y\_MOUNTORIENTATION\_NINE**) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get\_mountPosition()****YRefFrame****refframe→mountPosition()****refframe.get\_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

```
function get_mountPosition( ) As Y_MOUNTPOSITION
```

**Retourne :**

une valeur parmi l'énumération `Y_MOUNTPOSITION` (`Y_MOUNTPOSITION_BOTTOM`, `Y_MOUNTPOSITION_TOP`, `Y_MOUNTPOSITION_FRONT`, `Y_MOUNTPOSITION_RIGHT`, `Y_MOUNTPOSITION_REAR`, `Y_MOUNTPOSITION_LEFT`), correspondant à l'installation dans une boîte, sur l'une des six faces

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→get(userData)****YRefFrame****refframe→userData()refframe.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**refframe→isOnline()|refframe.isOnline()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du référentiel sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le référentiel est joignable, false sinon

**refframe→load()refframe.load()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→more3DCalibration()**  
**refframe.more3DCalibration()****YRefFrame**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.

**function more3DCalibration( ) As Integer**

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode get\_3DCalibrationHint (les instructions changent pendant la procédure de calibration). En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→nextRefFrame()refframe.nextRefFrame()****YRefFrame**

Continue l'énumération des référentiels commencée à l'aide de `yFirstRefFrame()`.

```
function nextRefFrame( ) As YRefFrame
```

**Retourne :**

un pointeur sur un objet `YRefFrame` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**refframe→registerValueCallback()**  
**refframe.registerValueCallback()****YRefFrame**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**refframe→save3DCalibration()**  
**refframe.save3DCalibration()****YRefFrame**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

```
function save3DCalibration( ) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après le redémarrage du module. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**refframe→set\_bearing()** **YRefFrame**  
**refframe→setBearing()refframe.set\_bearing()**

---

Modifie le cap de référence utilisé par le compas.

```
function set_bearing( ByVal newval As Double) As Integer
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici. Par exemple, si vous indiquez comme cap de référence la valeur de la déclinaison magnétique terrestre, le compas donnera l'orientation par rapport au Nord géographique. De même, si le capteur n'est pas positionné dans une des directions standard à cause d'un angle de lacet supplémentaire, vous pouvez le configurer comme cap de référence afin que le compas donne la direction naturelle attendue.

N'oubliez pas d'appeler la méthode saveToFlash( ) du module si le réglage doit être préservé.

**Paramètres :**

**newval** une valeur numérique représentant le cap de référence utilisé par le compas

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_logicalName()**  
**refframe→setLogicalName()**  
**refframe.set\_logicalName()**

**YRefFrame**

Modifie le nom logique du référentiel.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du référentiel.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**refframe→set\_mountPosition()**  
**refframe→setMountPosition()**  
**refframe.set\_mountPosition()**

**YRefFrame**

Modifie le référentiel de la boussole et des inclinomètres.

**function set\_mountPosition( ) As Integer**

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas où<sup>1</sup> 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).

```
procedure set(userData( ByVal data As Object)
```

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()****YRefFrame**

Initie le processus de calibration tridimensionnelle des capteurs.

```
function start3DCalibration( ) As Integer
```

Cette calibration est utilisée à bas niveau pour l'estimation innertielle de position et pour améliorer la précision des mesures d'inclinaison. Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.38. 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)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**relay→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YRelay.FindRelay() yFindRelay()yFindRelay()

YRelay

Permet de retrouver un relais d'après un identifiant donné.

```
function yFindRelay( ByVal func As String) As YRelay
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le relais soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRelay.isOnLine()` pour tester si le relais est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le relais sans ambiguïté

### Retourne :

un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

## YRelay.FirstRelay() yFirstRelay()yFirstRelay()

**YRelay**

Commence l'énumération des relais accessibles par la librairie.

```
function yFirstRelay( ) As YRelay
```

Utiliser la fonction `YRelay.nextRelay()` pour itérer sur les autres relais.

**Retourne :**

un pointeur sur un objet `YRelay`, correspondant au premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

**relay→delayedPulse()relay.delayedPulse()**

YRelay

Pré-programme une impulsion

```
function delayedPulse( ByVal ms_delay As Integer,  
                      ByVal ms_duration As Integer) As Integer
```

**Paramètres :**

**ms\_delay**      délai d'attente avant l'impulsion, en millisecondes

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## relay→describe()relay.describe()

## YRelay

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le relais (ex: Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**relay→get\_advertisedValue()**

**YRelay**

**relay→advertisedValue()relay.get\_advertisedValue()**

---

Retourne la valeur courante du relais (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du relais (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

**relay→get\_countdown()****YRelay****relay→countdown()relay.get\_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

```
function get_countdown( ) As Long
```

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne Y\_COUNTDOWN\_INVALID.

**relay→getErrorMessage()**

YRelay

**relay→errorMessage()relay.getErrorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du relais.

**relay→get\_errorType()****YRelay****relay→errorType()relay.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**relay→functionDescriptor()**  
**relay.get\_functionDescriptor()**

---

YRelay

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**relay→get\_functionId()****YRelay****relay→functionId()relay.get\_functionId()**

Retourne l'identifiant matériel du relais, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant le relais (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

<b>relay→get.hardwareId()</b>	<b>YRelay</b>
<b>relay→hardwareId()relay.get.hardwareId()</b>	

---

Retourne l'identifiant matériel unique du relais au format SERIAL.FUNCTIONID.

```
function get.hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du relais (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le relais (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**relay→get\_logicalName()****YRelay****relay→logicalName()relay.get\_logicalName()**

Retourne le nom logique du relais.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du relais.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**relay→get\_maxTimeOnStateA()**  
**relay→maxTimeOnStateA()**  
**relay.get\_maxTimeOnStateA()**

YRelay

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

function **get\_maxTimeOnStateA( ) As Long**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEA\_INVALID.

**relay→get\_maxTimeOnStateB()**  
**relay→maxTimeOnStateB()**  
**relay.get\_maxTimeOnStateB()**

**YRelay**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

function **get\_maxTimeOnStateB( ) As Long**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne **Y\_MAXTIMEONSTATEB\_INVALID**.

**relay→get\_module()**

**YRelay**

**relay→module()relay.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**relay→get\_output()****YRelay****relay→output()relay.get\_output()**

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
function get_output( ) As Integer
```

**Retourne :**

soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUT\_INVALID.

---

<b>relay→get_pulseTimer()</b>	<b>YRelay</b>
<b>relay→pulseTimer()relay.get_pulseTimer()</b>	

---

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
function get_pulseTimer( ) As Long
```

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.

---

<b>relay→get_state()</b>	<b>YRelay</b>
<b>relay→state()relay.get_state()</b>	

---

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

```
function get_state( ) As Integer
```

**Retourne :**

soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y\_STATE\_INVALID.

---

<b>relay→get_stateAtPowerOn()</b>	<b>YRelay</b>
<b>relay→stateAtPowerOn()relay.get_stateAtPowerOn()</b>	

---

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function get_stateAtPowerOn( ) As Integer
```

**Retourne :**

une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B représentant l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y\_STATEATPOWERON\_INVALID.

**relay→get(userData)****YRelay****relay→userData()relay.get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**relay→isOnline()relay.isOnline()****YRelay**

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le relais est joignable, false sinon

## relay→load()relay.load()

## YRelay

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

### Paramètres :

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **relay→nextRelay()|relay.nextRelay()**

**YRelay**

---

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

```
function nextRelay( ) As YRelay
```

**Retourne :**

un pointeur sur un objet `YRelay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**relay→pulse()**relay.pulse()******YRelay**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
function pulse( ByVal ms_duration As Integer) As Integer
```

**Paramètres :**

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→registerValueCallback()**  
**relay.registerValueCallback()****YRelay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**relay→set\_logicalName()****YRelay****relay→setLogicalName()relay.set\_logicalName()**

Modifie le nom logique du relais.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du relais.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_maxTimeOnStateA()**  
**relay→setMaxTimeOnStateA()**  
**relay.set\_maxTimeOnStateA()**

YRelay

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
function set_maxTimeOnStateA( ByVal newval As Long) As Integer
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_maxTimeOnStateB()**  
**relay→setMaxTimeOnStateB()**  
**relay.set\_maxTimeOnStateB()**

**YRelay**

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
function set_maxTimeOnStateB( ByVal newval As Long) As Integer
```

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay→set\_output()****YRelay****relay→setOutput()relay.set\_output()**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
function set_output( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**relay->set\_state()****YRelay****relay->setState()relay.set\_state()**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
function set_state( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>relay→set_stateAtPowerOn()</b>	<b>YRelay</b>
<b>relay→setStateAtPowerOn()</b>	
<b>relay.set_stateAtPowerOn()</b>	

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function set_stateAtPowerOn( ByVal newval As Integer) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.39. Interface des fonctions de type senseur

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_api.js'></script>
nodejs var yoctolib = require('yoctolib');
var YAPI = yoctolib.YAPI;
var YModule = yoctolib.YModule;
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #import "yocto_api.h"
pas uses yocto_api;
vb yocto_api.vb
cs yocto_api.cs
java import com.yoctopuce.YoctoAPI.YModule;
py from yocto_api import *

```

### Fonction globales

#### **yFindSensor(func)**

Permet de retrouver un senseur d'après un identifiant donné.

#### **yFirstSensor()**

Commence l'énumération des senseurs accessibles par la librairie.

### Méthodes des objets YSensor

#### **sensor->calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **sensor->describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format TYPE(NAME)=SERIAL.FUNCTIONID.

#### **sensor->get\_advertisedValue()**

Retourne la valeur courante du senseur (pas plus de 6 caractères).

#### **sensor->get\_currentRawValue()**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

#### **sensor->get\_currentValue()**

Retourne la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

#### **sensor->get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

#### **sensor->get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

#### **sensor->get\_friendlyName()**

Retourne un identifiant global du senseur au format NOM\_MODULE.NOM\_FONCTION.

#### **sensor->get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **sensor->get\_functionId()**

Retourne l'identifiant matériel du senseur, sans référence au module.

#### **sensor->get\_hardwareId()**

	Retourne l'identifiant matériel unique du senseur au format SERIAL . FUNCTIONID.
<b>sensor→get_highestValue()</b>	Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.
<b>sensor→get_logFrequency()</b>	Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.
<b>sensor→get_logicalName()</b>	Retourne le nom logique du senseur.
<b>sensor→get_lowestValue()</b>	Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.
<b>sensor→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>sensor→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>sensor→get_recordedData(startTime, endTime)</b>	Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.
<b>sensor→get_reportFrequency()</b>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<b>sensor→get_resolution()</b>	Retourne la résolution des valeurs mesurées.
<b>sensor→get_unit()</b>	Retourne l'unité dans laquelle la mesure est exprimée.
<b>sensor→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>sensor→isOnline()</b>	Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.
<b>sensor→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.
<b>sensor→load(msValidity)</b>	Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.
<b>sensor→loadCalibrationPoints(rawValues, refValues)</b>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.
<b>sensor→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.
<b>sensor→nextSensor()</b>	Continue l'énumération des senseurs commencée à l'aide de yFirstSensor( ).
<b>sensor→registerTimedReportCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<b>sensor→registerValueCallback(callback)</b>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>sensor→set_highestValue(newval)</b>	Modifie la mémoire de valeur maximale observée.
<b>sensor→set_logFrequency(newval)</b>	

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**sensor→set\_logicalName(newval)**

Modifie le nom logique du senseur.

**sensor→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**sensor→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**sensor→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**sensor→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**sensor→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YSensor.FindSensor() yFindSensor()yFindSensor()

YSensor

Permet de retrouver un senseur d'après un identifiant donné.

```
function yFindSensor( ByVal func As String) As YSensor
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le senseur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSensor.isOnline()` pour tester si le senseur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le senseur sans ambiguïté

### Retourne :

un objet de classe `YSensor` qui permet ensuite de contrôler le senseur.

## YSensor.FirstSensor() yFirstSensor()yFirstSensor()

YSensor

Commence l'énumération des senseurs accessibles par la librairie.

```
function yFirstSensor( ) As YSensor
```

Utiliser la fonction `YSensor.nextSensor()` pour itérer sur les autres senseurs.

**Retourne :**

un pointeur sur un objet `YSensor`, correspondant au premier senseur accessible en ligne, ou `null` si il n'y a pas de senseurs disponibles.

**sensor→calibrateFromPoints()**  
**sensor.calibrateFromPoints()****YSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→describe()sensor.describe()****YSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le senseur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**sensor→get\_advertisedValue()**  
**sensor→advertisedValue()**  
**sensor.get\_advertisedValue()**

**YSensor**

Retourne la valeur courante du senseur (pas plus de 6 caractères).

function **get\_advertisedValue( ) As String**

**Retourne :**

une chaîne de caractères représentant la valeur courante du senseur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne **Y\_ADVERTISEDVALUE\_INVALID**.

**sensor→get\_currentRawValue()**  
**sensor→currentRawValue()**  
**sensor.get\_currentRawValue()**

**YSensor**

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTRAWVALUE\_INVALID**.

---

<b>sensor→get_currentValue()</b>	<b>YSensor</b>
<b>sensor→currentValue()sensor.get_currentValue()</b>	

---

Retourne la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**sensor→get\_errorMessage()** YSensor  
**sensor→errorMessage()sensor.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

---

**sensor→get\_errorType()****YSensor****sensor→errorType()sensor.get\_errorType()**

---

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**sensor→functionDescriptor()**  
**sensor.get\_functionDescriptor()**

**YSensor**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**sensor→get\_functionId()****YSensor****sensor→functionId()sensor.get\_functionId()**

Retourne l'identifiant matériel du senseur, sans référence au module.

```
function get_functionId( ) As String
```

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.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du senseur (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le senseur (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

<b>sensor→get_highestValue()</b>	<b>YSensor</b>
<b>sensor→highestValue()sensor.get_highestValue()</b>	

---

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

**sensor→get\_logFrequency()**

**YSensor**

**sensor→logFrequency()sensor.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**sensor→get\_logicalName()****YSensor****sensor→logicalName()sensor.get\_logicalName()**

Retourne le nom logique du senseur.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du senseur.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**sensor→get\_lowestValue()** YSensor  
**sensor→lowestValue()sensor.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

function **get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

---

**sensor→get\_module()****YSensor****sensor→module()sensor.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

<b>sensor→get_recordedData()</b>	<b>YSensor</b>
<b>sensor→recordedData()sensor.get_recordedData()</b>	

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**sensor→get\_reportFrequency()**  
**sensor→reportFrequency()**  
**sensor.get\_reportFrequency()**

**YSensor**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

function **get\_reportFrequency( ) As String**

**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.

**function get\_resolution( ) As Double**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

---

**sensor→get\_unit()****YSensor****sensor→unit()sensor.get\_unit()**

---

Retourne l'unité dans laquelle la mesure est exprimée.function **get\_unit( ) As String****Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**sensor→get(userData)**

**YSensor**

**sensor→userData()sensor.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**function get(userData) As Object**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**sensor→isOnline()sensor.isOnline()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du senseur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le senseur est joignable, false sinon

**sensor→load()sensor.load()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

```
procedure loadCalibrationPoints( )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **sensor→nextSensor()sensor.nextSensor()**

**YSensor**

---

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

```
function nextSensor( ) As YSensor
```

**Retourne :**

un pointeur sur un objet `YSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**sensor→registerTimedReportCallback()**  
**sensor.registerTimedReportCallback()****YSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

**sensor→registerValueCallback()  
sensor.registerValueCallback()****YSensor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

---

**sensor→set\_highestValue()**  
**sensor→setHighestValue()sensor.set\_highestValue()****YSensor**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_logFrequency()**  
**sensor→setLogFrequency()**  
**sensor.set\_logFrequency()**

**YSensor**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>sensor→set_logicalName()</b>	<b>YSensor</b>
<b>sensor→setLogicalName()sensor.set_logicalName()</b>	

---

Modifie le nom logique du senseur.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**sensor→set\_reportFrequency()**  
**sensor→setReportFrequency()**  
**sensor.set\_reportFrequency()**

**YSensor**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>sensor→set_resolution()</b>	<b>YSensor</b>
<b>sensor→setResolution()sensor.set_resolution()</b>	

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.40. Interface de la fonction SerialPort

La fonction SerialPort permet de piloter entièrement un module d'interface série Yoctopuce, pour envoyer et recevoir des données et configurer les paramètres de transmission (vitesse, nombre de bits, parité, contrôle de flux et protocole). Notez que les interfaces série Yoctopuce ne sont pas des visibles comme des ports COM virtuels. Ils sont faits pour être utilisés comme tous les autres modules Yoctopuce.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_serialport.js'></script>
nodejs var yoctolib = require('yoctolib');
var YSerialPort = yoctolib.YSerialPort;
php require_once('yocto_serialport.php');
cpp #include "yocto_serialport.h"
m #import "yocto_serialport.h"
pas uses yocto_serialport;
vb yocto_serialport.vb
cs yocto_serialport.cs
java import com.yoctopuce.YoctoAPI.YSerialPort;
py from yocto_serialport import *

```

### Fonction globales

#### yFindSerialPort(func)

Permet de retrouver une port série d'après un identifiant donné.

#### yFirstSerialPort()

Commence l'énumération des le port série accessibles par la librairie.

### Méthodes des objets YSerialPort

#### serialport→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port série au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### serialport→get\_CTS()

Lit l'état de la ligne CTS.

#### serialport→get\_advertisedValue()

Retourne la valeur courante du port série (pas plus de 6 caractères).

#### serialport→get\_errCount()

Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.

#### serialport→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port série.

#### serialport→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port série.

#### serialport→get\_friendlyName()

Retourne un identifiant global du port série au format NOM\_MODULE . NOM\_FONCTION.

#### serialport→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### serialport→get\_functionId()

Retourne l'identifiant matériel du port série, sans référence au module.

#### serialport→get\_hardwareId()

Retourne l'identifiant matériel unique du port série au format SERIAL . FUNCTIONID.

<b>serialport→get_lastMsg()</b>	Retourne le dernier message reçu (pour les protocoles de type Line, Frame et Modbus).
<b>serialport→get_logicalName()</b>	Retourne le nom logique du port série.
<b>serialport→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>serialport→get_module_async(callback, context)</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>serialport→get_msgCount()</b>	Retourne le nombre de messages reçus depuis la dernière mise à zéro.
<b>serialport→get_protocol()</b>	Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.
<b>serialport→get_rxCount()</b>	Retourne le nombre d'octets reçus depuis la dernière mise à zéro.
<b>serialport→get_serialMode()</b>	Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".
<b>serialport→get_txCount()</b>	Retourne le nombre d'octets transmis depuis la dernière mise à zéro.
<b>serialport→get(userData)</b>	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>serialport→isOnline()</b>	Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.
<b>serialport→isOnline_async(callback, context)</b>	Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.
<b>serialport→load(msValidity)</b>	Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.
<b>serialport→load_async(msValidity, callback, context)</b>	Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.
<b>serialport→modbusReadBits(slaveNo, pduAddr, nBits)</b>	Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.
<b>serialport→modbusReadInputBits(slaveNo, pduAddr, nBits)</b>	Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.
<b>serialport→modbusReadInputRegisters(slaveNo, pduAddr, nWords)</b>	Lit un ou plusieurs registres d'entrée (registre en lecture seule) depuis un périphérique MODBUS.
<b>serialport→modbusReadRegisters(slaveNo, pduAddr, nWords)</b>	Lit un ou plusieurs registres interne depuis un périphérique MODBUS.
<b>serialport→modbusWriteAndReadRegisters(slaveNo, pduWriteAddr, values, pduReadAddr, nReadWords)</b>	Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.
<b>serialport→modbusWriteBit(slaveNo, pduAddr, value)</b>	Modifie l'état d'un seul bit (ou relais) sur un périphérique MODBUS.
<b>serialport→modbusWriteBits(slaveNo, pduAddr, bits)</b>	Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.
<b>serialport→modbusWriteRegister(slaveNo, pduAddr, value)</b>	Modifie la valeur d'un registre interne 16 bits sur un périphérique MODBUS.
<b>serialport→modbusWriteRegisters(slaveNo, pduAddr, values)</b>	

### 3. Reference

Modifie l'état de plusieurs registres internes 16 bits contigus sur un périphérique MODBUS.
<b>serialport→nextSerialPort()</b> Continue l'énumération des le port série commencée à l'aide de <code>yFirstSerialPort()</code> .
<b>serialport→queryLine(query, maxWait)</b> Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.
<b>serialport→queryMODBUS(slaveNo, pduBytes)</b> Envoie un message à un périphérique MODBUS esclave connecté au port série, et lit la réponse reçue.
<b>serialport→readHex(nBytes)</b> Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.
<b>serialport→readLine()</b> Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.
<b>serialport→readMessages(pattern, maxWait)</b> Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.
<b>serialport→readStr(nChars)</b> Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.
<b>serialport→read_seek(rxCountVal)</b> Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.
<b>serialport→registerValueCallback(callback)</b> Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>serialport→reset()</b> Remet à zéro tous les compteurs et efface les tampons.
<b>serialport→set_RTS(val)</b> Change manuellement l'état de la ligne RTS.
<b>serialport→set_logicalName(newval)</b> Modifie le nom logique du port série.
<b>serialport→set_protocol(newval)</b> Modifie le type de protocol utilisé sur la communication série.
<b>serialport→set_serialMode(newval)</b> Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".
<b>serialport→set(userData)</b> Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get(userData)</code> .
<b>serialport→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.
<b>serialport→writeArray(byteList)</b> Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.
<b>serialport→writeBin(buff)</b> Envoie un objet binaire tel quel sur le port série.
<b>serialport→writeHex(hexString)</b> Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.
<b>serialport→writeLine(text)</b>

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

**serialport→writeMODBUS(hexString)**

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

**serialport→writeStr(text)**

Envoie une chaîne de caractères telle quelle sur le port série.

## YSerialPort.FindSerialPort() yFindSerialPort()yFindSerialPort()

YSerialPort

Permet de retrouver une port série d'après un identifiant donné.

```
function yFindSerialPort( ByVal func As String) As YSerialPort
```

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 série soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSerialPort.isOnline()` pour tester si le port série 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 série sans ambiguïté

### Retourne :

un objet de classe `YSerialPort` qui permet ensuite de contrôler le port série.

## YSerialPort.FirstSerialPort() yFirstSerialPort()yFirstSerialPort()

## YSerialPort

Commence l'énumération des le port série accessibles par la librairie.

```
function yFirstSerialPort( ) As YSerialPort
```

Utiliser la fonction YSerialPort.nextSerialPort( ) pour itérer sur les autres le port série.

### Retourne :

un pointeur sur un objet YSerialPort, correspondant au premier port série accessible en ligne, ou null si il n'y a pas du port série disponibles.

**serialport→describe()serialport.describe()****YSerialPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port série au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe( ) As String
```

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 série (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**serialport→get\_CTS()****YSerialPort****serialport→CTS()serialport.get\_CTS()**

Lit l'état de la ligne CTS.

```
function get_CTS( ) As Integer
```

La ligne CTS est habituellement pilotée par le signal RTS du périphérique série connecté.

**Retourne :**

1 si le CTS est signalé (niveau haut), 0 si le CTS n'est pas actif (niveau bas).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→get\_advertisedValue()**  
**serialport→advertisedValue()**  
**serialport.get\_advertisedValue()**

---

**YSerialPort**

Retourne la valeur courante du port série (pas plus de 6 caractères).

**function get\_advertisedValue( ) As String**

**Retourne :**

une chaîne de caractères représentant la valeur courante du port série (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVISEDVALUE\_INVALID.

**serialport→get\_errCount()****YSerialPort****serialport→errCount()serialport.get\_errCount()**

Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.

```
function get_errCount( ) As Integer
```

**Retourne :**

un entier représentant le nombre d'erreurs de communication détectées depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne Y\_ERRCOUNT\_INVALID.

**serialport→get\_errorMessage()**  
**serialport→errorMessage()**  
**serialport.get\_errorMessage()**

**YSerialPort**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port série.

**function get\_errorMessage( ) As String**

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 série.

**serialport→get\_errorType()****YSerialPort****serialport→errorType()serialport.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port série.

```
function get_errorType( ) As YRETCODE
```

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 série.

**serialport→get\_functionDescriptor()**  
**serialport→functionDescriptor()**  
**serialport.get\_functionDescriptor()**

---

**YSerialPort**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

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

**serialport→get\_functionId()****YSerialPort****serialport→functionId()serialport.get\_functionId()**

Retourne l'identifiant matériel du port série, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le port série (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**serialport→get\_hardwareId()**

**YSerialPort**

**serialport→hardwareId()serialport.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du port série au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port série (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le port série (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**serialport→get\_lastMsg()****YSerialPort****serialport→lastMsg()serialport.get\_lastMsg()**

Retourne le dernier message reçu (pour les protocoles de type Line, Frame et Modbus).

```
function get_lastMsg( ) As String
```

**Retourne :**

une chaîne de caractères représentant le dernier message reçu (pour les protocoles de type Line, Frame et Modbus)

En cas d'erreur, déclenche une exception ou retourne Y\_LASTMSG\_INVALID.

**serialport→get\_logicalName()**  
**serialport→logicalName()**  
**serialport.get\_logicalName()**

**YSerialPort**

Retourne le nom logique du port série.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du port série.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**serialport→get\_module()****YSerialPort****serialport→module()serialport.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**serialport→get\_msgCount()**

**YSerialPort**

**serialport→msgCount()serialport.get\_msgCount()**

---

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

```
function get_msgCount( ) As Integer
```

**Retourne :**

un entier représentant le nombre de messages reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne Y\_MSGCOUNT\_INVALID.

**serialport→get\_protocol()****YSerialPort****serialport→protocol()serialport.get\_protocol()**

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

```
function get_protocol( ) As String
```

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Modbus-ASCII" pour des messages MODBUS en mode ASCII, "Modbus-RTU" pour des messages MODBUS en mode RTU, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

**Retourne :**

une chaîne de caractères représentant le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères

En cas d'erreur, déclenche une exception ou retourne Y\_PROTOCOL\_INVALID.

**serialport→get\_rxCount()**

**YSerialPort**

**serialport→rxCount()serialport.get\_rxCount()**

---

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

**function get\_rxCount( ) As Integer**

**Retourne :**

un entier représentant le nombre d'octets reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne Y\_RXCOUNT\_INVALID.

**serialport→get\_serialMode()****YSerialPort****serialport→serialMode()serialport.get\_serialMode()**

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

```
function get_serialMode( ) As String
```

La chaîne contient le taux de transfert, le nombre de bits de données, la parité parité et le nombre de bits d'arrêt. Un suffixe supplémentaire optionnel est inclus si une option de contrôle de flux est active: "CtsRts" pour le contrôle de flux matériel, "XOnXOff" pour le contrôle de flux logique et "Simplex" pour l'utilisation du signal RTS pour l'acquisition d'un bus partagé (tel qu'utilisé pour certains adaptateurs RS485 par exemple).

**Retourne :**

une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1"

En cas d'erreur, déclenche une exception ou retourne **Y\_SERIALMODE\_INVALID**.

**serialport→get\_txCount()**

**YSerialPort**

**serialport→txCount()serialport.get\_txCount()**

---

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

**function get\_txCount( ) As Integer**

**Retourne :**

un entier représentant le nombre d'octets transmis depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne Y\_TCOUNT\_INVALID.

**serialport→get(userData)****YSerialPort****serialport→userData()serialport.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

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.

**serialport→isOnline()|serialport.isOnline()****YSerialPort**

Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du port série sont valides au moment de l'appel, le module est considéré joignable. 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 série est joignable, false sinon

**serialport→load()serialport.load()****YSerialPort**

Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

## serialport→modbusReadBits() serialport.modbusReadBits()

YSerialPort

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

```
function modbusReadBits( ) As List
```

Cette méthode utilise le code de fonction MODBUS 0x01 (Read Coils).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à interroger

**pduAddr** adresse relative du premier bit à lire (indexé à partir de zéro).

**nBits** nombre de bits à lire

**Retourne :**

un vecteur d'entiers, correspondant chacun à un bit.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**serialport→modbusReadInputBits()  
serialport.modbusReadInputBits()****YSerialPort**

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

function **modbusReadInputBits( ) As List**

Cette méthode utilise le code de fonction MODBUS 0x02 (Read Discrete Inputs).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à interroger

**pduAddr** adresse relative du premier bit à lire (indexé à partir de zéro).

**nBits** nombre de bits à lire

**Retourne :**

un vecteur d'entiers, correspondant chacun à un bit.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**serialport→modbusReadInputRegisters()**  
**serialport.modbusReadInputRegisters()****YSerialPort**

Lit un ou plusieurs registres d'entrée (registre en lecture seule) depuis un périphérique MODBUS.

**function modbusReadInputRegisters( ) As List**

Cette méthode utilise le code de fonction MODBUS 0x04 (Read Input Registers).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à interroger

**pduAddr** adresse relative du premier registre d'entrée à lire (indexé à partir de zéro).

**nWords** nombre de registres d'entrée à lire

**Retourne :**

un vecteur d'entiers, correspondant chacun à une valeur d'entrée (16 bits).

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**serialport→modbusReadRegisters()**  
**serialport.modbusReadRegisters()****YSerialPort**

Lit un ou plusieurs registres interne depuis un périphérique MODBUS.

function **modbusReadRegisters( ) As List**

Cette méthode utilise le code de fonction MODBUS 0x03 (Read Holding Registers).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à interroger

**pduAddr** adresse relative du premier registre interne à lire (indexé à partir de zéro).

**nWords** nombre de registres internes à lire

**Retourne :**

un vecteur d'entiers, correspondant chacun à une valeur de registre (16 bits).

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**serialport→modbusWriteAndReadRegisters()**  
**serialport.modbusWriteAndReadRegisters()****YSerialPort**

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

**procedure modbusWriteAndReadRegisters( )**

Cette méthode utilise le code de fonction MODBUS 0x17 (Read/Write Multiple Registers).

**Paramètres :**

**slaveNo**      adresse du périphérique MODBUS esclave à piloter

**pduWriteAddr**    adresse relative du premier registre interne à modifier (indexé à partir de zéro).

**values**        vecteur de valeurs 16 bits à appliquer

**pduReadAddr**    adresse relative du premier registre interne à lire (indexé à partir de zéro).

**nReadWords**    nombre de registres internes à lire

**Retourne :**

un vecteur d'entiers, correspondant chacun à une valeur de registre (16 bits) lue.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**serialport→modbusWriteBit()  
serialport.modbusWriteBit()****YSerialPort**

Modifie l'état d'un seul bit (ou relais) sur un périphérique MODBUS.

```
function modbusWriteBit( ) As Integer
```

Cette méthode utilise le code de fonction MODBUS 0x05 (Write Single Coil).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à piloter

**pduAddr** adresse relative du bit à modifier (indexé à partir de zéro).

**value** la valeur à appliquer (0 pour l'état OFF, non-zéro pour l'état ON)

**Retourne :**

le nombre de bits affectés sur le périphérique (1)

En cas d'erreur, déclenche une exception ou retourne zéro.

## **serialport→modbusWriteBits()** **serialport.modbusWriteBits()**

**YSerialPort**

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

**procedure modbusWriteBits( )**

Cette méthode utilise le code de fonction MODBUS 0x0f (Write Multiple Coils).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à piloter

**pduAddr** adresse relative du premier bit à modifier (indexé à partir de zéro).

**bits** vecteur de bits à appliquer (un entier par bit)

**Retourne :**

le nombre de bits affectés sur le périphérique

En cas d'erreur, déclenche une exception ou retourne zéro.

**serialport→modbusWriteRegister()  
serialport.modbusWriteRegister()****YSerialPort**

Modifie la valeur d'un registre interne 16 bits sur un périphérique MODBUS.

function **modbusWriteRegister( ) As Integer**

Cette méthode utilise le code de fonction MODBUS 0x06 (Write Single Register).

**Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à piloter

**pduAddr** adresse relative du registre à modifier (indexé à partir de zéro).

**value** la valeur 16 bits à appliquer

**Retourne :**

le nombre de registres affectés sur le périphérique (1)

En cas d'erreur, déclenche une exception ou retourne zéro.

## **serialport→modbusWriteRegisters()** **serialport.modbusWriteRegisters()**

**YSerialPort**

Modifie l'état de plusieurs registres internes 16 bits contigus sur un périphérique MODBUS.

**procedure modbusWriteRegisters( )**

Cette méthode utilise le code de fonction MODBUS 0x10 (Write Multiple Registers).

### **Paramètres :**

**slaveNo** adresse du périphérique MODBUS esclave à piloter

**pduAddr** adresse relative du premier registre interne à modifier (indexé à partir de zéro).

**values** vecteur de valeurs 16 bits à appliquer

### **Retourne :**

le nombre de registres affectés sur le périphérique

En cas d'erreur, déclenche une exception ou retourne zéro.

**serialport→nextSerialPort()serialport.nextSerialPort()****YSerialPort**

Continue l'énumération des le port série commencée à l'aide de `yFirstSerialPort()`.

```
function nextSerialPort() As YSerialPort
```

**Retourne :**

un pointeur sur un objet `YSerialPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**serialport→queryLine()|serialport.queryLine()****YSerialPort**

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

**function queryLine( ) As String**

Cette fonction ne peut être utilisée que lorsque le module est configuré en protocole 'Line'.

**Paramètres :**

**query** le message à envoyer (sans le retour de chariot)

**maxWait** le temps maximum d'attente pour obtenir une réponse (en millisecondes).

**Retourne :**

la première ligne de texte reçue après l'envoi du message. Les lignes suivantes peuvent être obtenues avec des appels à readLine ou readMessages.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→queryMODBUS()**  
**serialport.queryMODBUS()****YSerialPort**

Envoie un message à un périphérique MODBUS esclave connecté au port série, et lit la réponse reçue.

**procedure queryMODBUS( )**

Le contenu du message est le PDU, fourni sous forme de vecteur d'octets.

**Paramètres :****slaveNo** adresse du périphérique MODBUS esclave**pduBytes** message à envoyer (PDU), sous forme de vecteur d'octets. Le premier octet du PDU est le code de fonction MODBUS.**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un tableau vide (ou une réponse d'erreur).

**serialport→readHex()|serialport.readHex()****YSerialPort**

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

```
function readHex( ) As String
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

**Paramètres :**

**nBytes** le nombre maximal d'octets à lire

**Retourne :**

une chaîne de caractère avec le contenu du tampon de réception, encodé en hexadécimal

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→readLine()serialport.readLine()****YSerialPort**

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

function **readLine( )** As String

Cette fonction ne peut être utilisée que lorsque le module est configuré pour gérer un protocole basé message, comme en mode 'Line' ou en protocole MODBUS. Elle ne fonctionne pas dans les modes de flux continu ('Char' et 'Byte'), pour lesquels le début d'un message n'est pas défini.

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction retournera la plus ancienne ligne disponible et déplacera le pointeur de position juste après. Si aucune nouvelle ligne entière n'est disponible, la fonction retourne un chaîne vide.

**Retourne :**

une chaîne de caractère avec une ligne de texte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→readMessages()  
serialport.readMessages()****YSerialPort**

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

**function readMessages( ) As List**

Cette fonction ne peut être utilisée que lorsque le module est configuré pour gérer un protocole basé message, comme en mode 'Line' ou en protocole MODBUS. Elle ne fonctionne pas dans les modes de flux continu ('Char' et 'Byte'), pour lesquels le début d'un message n'est pas défini.

La recherche retourne tous les messages trouvés qui correspondent au format. Tant qu'aucun message adéquat n'est trouvé, la fonction attendra, au maximum pour le temps spécifié en argument (en millisecondes).

**Paramètres :**

**pattern** une expression régulière limitée décrivant le format de message désiré, ou une chaîne vide si aucun filtrage des messages n'est désiré. Pour les protocoles binaires, le format est appliqué à la représentation hexadécimale du message.

**maxWait** le temps maximum d'attente pour obtenir un message, tant qu'aucun n'est trouvé dans le tampon de réception (en millisecondes).

**Retourne :**

un tableau de chaînes de caractères contenant les messages trouvés. Les messages binaires sont convertis automatiquement en représentation hexadécimale.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

**serialport→readStr()serialport.readStr()****YSerialPort**

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

```
function readStr( ) As String
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

**Paramètres :**

**nChars** le nombre maximum de caractères à lire

**Retourne :**

une chaîne de caractère avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## serialport→read\_seek()serialport.read\_seek()

## YSerialPort

---

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

```
function read_seek( ) As Integer
```

Cette fonction n'a pas d'effet sur le module, elle ne fait que changer la valeur stockée dans l'objet YSerialPort qui sera utilisée pour les prochaines opérations de lecture.

**Paramètres :**

**rxCountVal** l'index de position absolue (valeur de rxCount) pour les opérations de lecture suivantes.

**Retourne :**

rien du tout.

**serialport→registerValueCallback()  
serialport.registerValueCallback()****YSerialPort**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

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.

## serialport→reset()serialport.reset()

YSerialPort

Remet à zéro tous les compteurs et efface les tampons.

```
function reset( ) As Integer
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→set\_RTS()****YSerialPort****serialport→setRTS()serialport.set\_RTS()**

Change manuellement l'état de la ligne RTS.

```
function set_RTS( ) As Integer
```

Cette fonction n'a pas d'effet lorsque le contrôle du flux par CTS/RTS est actif, car la ligne RTS est alors pilotée automatiquement.

**Paramètres :**

**val** 1 pour activer la ligne RTS, 0 pour la 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.

**serialport→set\_logicalName()**  
**serialport→setLogicalName()**  
**serialport.set\_logicalName()**

**YSerialPort**

Modifie le nom logique du port série.

**function set\_logicalName( ByVal newval As String) As Integer**

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 série.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→set\_protocol()****YSerialPort****serialport→setProtocol()serialport.set\_protocol()**

Modifie le type de protocol utilisé sur la communication série.

```
function set_protocol( ByVal newval As String) As Integer
```

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Modbus-ASCII" pour des messages MODBUS en mode ASCII, "Modbus-RTU" pour des messages MODBUS en mode RTU, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

**Paramètres :**

**newval** une chaîne de caractères représentant le type de protocol utilisé sur la communication série

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→set\_serialMode()**  
**serialport→setSerialMode()**  
**serialport.set\_serialMode()**

**YSerialPort**

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

function **set\_serialMode( ByVal newval As String) As Integer**

La chaîne contient le taux de transfert, le nombre de bits de données, la parité parité et le nombre de bits d'arrêt. Un suffixe supplémentaire optionnel peut être inclus pour activer une option de contrôle de flux: "CtsRts" pour le contrôle de flux matériel, "XOnXOff" pour le contrôle de flux logique et "Simplex" pour l'utilisation du signal RTS pour l'acquisition d'un bus partagé (tel qu'utilisé pour certains adaptateurs RS485 par exemple).

**Paramètres :**

**newval** une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1"

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→set(userData)****YSerialPort****serialport→setUserData()serialport.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).

```
procedure set(userData( ByVal data As Object)
```

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

## serialport→writeArray()serialport.writeArray()

YSerialPort

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

procedure **writeArray( )**

**Paramètres :**

**byteList** la liste d'octets à envoyer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→writeBin()serialport.writeBin()****YSerialPort**

Envoie un objet binaire tel quel sur le port série.

```
procedure writeBin( )
```

**Paramètres :**

**buff** l'objet binaire à envoyer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→writeHex()serialport.writeHex()****YSerialPort**

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

```
function writeHex( ) As Integer
```

**Paramètres :**

**hexString** la chaîne hexadécimale à envoyer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→writeLine()serialport.writeLine()****YSerialPort**

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

```
function writeLine( ) As Integer
```

**Paramètres :**

**text** la chaîne de caractères à envoyer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→writeMODBUS()****YSerialPort**

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

**function writeMODBUS( ) As Integer**

Le message doit commencer par l'adresse de destination. Le CRC (ou LRC) MODBUS est ajouté automatiquement par la fonction. Cette fonction n'attend pas de réponse.

**Paramètres :**

**hexString** le message à envoyer, en hexadécimal, sans le CRC/LRC

**Retourne :**

**YAPI\_SUCCESS** si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**serialport→writeStr()serialport.writeStr()****YSerialPort**

Envoie une chaîne de caractères telle quelle sur le port série.

```
function writeStr( ) As Integer
```

**Paramètres :**

**text** la chaîne de caractères à envoyer

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.41. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'intervalle de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_servo.js'></script>
nodejs var yoctolib = require('yoctolib');
var YServo = yoctolib.YServo;
php require_once('yocto_servo.php');
cpp #include "yocto_servo.h"
m #import "yocto_servo.h"
pas uses yocto_servo;
vb yocto_servo.vb
cs yocto_servo.cs
java import com.yoctopuce.YoctoAPI.YServo;
py from yocto_servo import *

```

### Fonction globales

#### **yFindServo(func)**

Permet de retrouver un servo d'après un identifiant donné.

#### **yFirstServo()**

Commence l'énumération des servo accessibles par la librairie.

### Méthodes des objets YServo

#### **servo→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format TYPE(NAME)=SERIAL.FUNCTIONID.

#### **servo→get\_advertisedValue()**

Retourne la valeur courante du servo (pas plus de 6 caractères).

#### **servo→get\_enabled()**

Retourne l'état de fonctionnement du \$FUNCTION\$.

#### **servo→get\_enabledAtPowerOn()**

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

#### **servo→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

#### **servo→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

#### **servo→get\_friendlyName()**

Retourne un identifiant global du servo au format NOM\_MODULE.NOM\_FONCTION.

#### **servo→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### **servo→get\_functionId()**

Retourne l'identifiant matériel du servo, sans référence au module.

#### **servo→get\_hardwareId()**

Retourne l'identifiant matériel unique du servo au format SERIAL.FUNCTIONID.

#### **servo→get\_logicalName()**

Retourne le nom logique du servo.

**`servo→get_module()`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`servo→get_module_async(callback, context)`**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**`servo→get_neutral()`**

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

**`servo→get_position()`**

Retourne la position courante du servo.

**`servo→get_positionAtPowerOn()`**

Retourne la position du servo au démarrage du module.

**`servo→get_range()`**

Retourne la plage d'utilisation du servo.

**`servo→get_userData()`**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set(userData)`.

**`servo→isOnline()`**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

**`servo→isOnline_async(callback, context)`**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

**`servo→load(msValidity)`**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

**`servo→load_async(msValidity, callback, context)`**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

**`servo→move(target, ms_duration)`**

Déclenche un mouvement à vitesse constante vers une position donnée.

**`servo→nextServo()`**

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

**`servo→registerValueCallback(callback)`**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**`servo→set_enabled(newval)`**

Démarre ou arrête le \$FUNCTION\$.

**`servo→set_enabledAtPowerOn(newval)`**

Configure l'état du générateur de signal de commande du servo au démarrage du module.

**`servo→set_logicalName(newval)`**

Modifie le nom logique du servo.

**`servo→set_neutral(newval)`**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

**`servo→set_position(newval)`**

Modifie immédiatement la consigne de position du servo.

**`servo→set_positionAtPowerOn(newval)`**

Configure la position du servo au démarrage du module.

**`servo→set_range(newval)`**

Modifie la plage d'utilisation du servo, en pourcents.

**`servo→set_userData(data)`**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get(userData)`.

**`servo→wait_async(callback, context)`**

### **3. Reference**

---

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YServo.FindServo() yFindServo()yFindServo()

YServo

Permet de retrouver un servo d'après un identifiant donné.

```
function yFindServo( ByVal func As String) As YServo
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le servo soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence le servo sans ambiguïté

### Retourne :

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

## YServo.FirstServo() yFirstServo()yFirstServo()

YServo

Commence l'énumération des servo accessibles par la librairie.

```
function yFirstServo( ) As YServo
```

Utiliser la fonction YServo.nextServo( ) pour itérer sur les autres servo.

**Retourne :**

un pointeur sur un objet YServo, correspondant au premier servo accessible en ligne, ou null si il n'y a pas de servo disponibles.

**servo→describe()servo.describe()****YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le servo (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**servo→get\_advertisedValue()** YServo  
**servo→advertisedValue()servo.get\_advertisedValue()**

---

Retourne la valeur courante du servo (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du servo (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**servo→get\_enabled()****YServo****servo→enabled()servo.get\_enabled()**

Retourne l'état de fonctionnement du \$FUNCTION\$.

```
function get_enabled( ) As Integer
```

**Retourne :**

soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE, selon l'état de fonctionnement du \$FUNCTION\$

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLED\_INVALID.

**servo→get\_enabledAtPowerOn()**  
**servo→enabledAtPowerOn()**  
**servo.get\_enabledAtPowerOn()**

**YServo**

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

function **get\_enabledAtPowerOn( ) As Integer**

**Retourne :**

soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_ENABLEDATPOWERON\_INVALID.

---

**servo→getErrorMessage()** YServo  
**servo→errorMessage()servo.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du servo.

**servo→get\_errorType()**  
**servo→errorType()servo.get\_errorType()**

---

**YServo**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

**function get\_errorType( ) As YRETCODE**

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\_functionDescriptor()**  
**servo→functionDescriptor()**  
**servo.get\_functionDescriptor()**

YServo

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**servo→get\_functionId()**

**YServo**

**servo→functionId()servo.get\_functionId()**

---

Retourne l'identifiant matériel du servo, sans référence au module.

**function get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le servo (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**servo→get\_hardwareId()****YServo****servo→hardwareId()servo.get\_hardwareId()**

Retourne l'identifiant matériel unique du servo au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du servo (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le servo (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**servo→get\_logicalName()**

**YServo**

**servo→logicalName()servo.get\_logicalName()**

---

Retourne le nom logique du servo.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du servo.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**servo→get\_module()****YServo****servo→module()servo.get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

**servo→get\_neutral()**

**YServo**

**servo→neutral()servo.get\_neutral()**

---

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

function **get\_neutral( ) As Integer**

**Retourne :**

un entier représentant la durée en microsecondes de l'impulsion correspondant au neutre du servo

En cas d'erreur, déclenche une exception ou retourne **Y\_NEUTRAL\_INVALID**.

**servo→get\_position()****YServo****servo→position()servo.get\_position()**

Retourne la position courante du servo.

```
function get_position( ) As Integer
```

**Retourne :**

un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne Y\_POSITION\_INVALID.

**servo→get\_positionAtPowerOn()**  
**servo→positionAtPowerOn()**  
**servo.get\_positionAtPowerOn()**

**YServo**

---

Retourne la position du servo au démarrage du module.

```
function get_positionAtPowerOn( ) As Integer
```

**Retourne :**

un entier représentant la position du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_POSITIONATPOWERON\_INVALID.

**servo→get\_range()****YServo****servo→range()servo.get\_range()**

Retourne la plage d'utilisation du servo.

```
function get_range( ) As Integer
```

**Retourne :**

un entier représentant la plage d'utilisation du servo

En cas d'erreur, déclenche une exception ou retourne Y\_RANGE\_INVALID.

**servo→get(userData)**

**YServo**

**servo→userData()servo.get(userData)**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**function get(userData) As Object**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**servo→isOnline()servo.isOnline()****YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du servo sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le servo est joignable, false sinon

**servo→load()servo.load()****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→move()servo.move()****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

```
function move( ByVal target As Integer,  
                ByVal ms_duration As Integer) As Integer
```

**Paramètres :**

**target** nouvelle position à la fin du mouvement

**ms\_duration** durée totale du mouvement, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **servo→nextServo()servo.nextServo()**

**YServo**

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

```
function nextServo( ) As YServo
```

**Retourne :**

un pointeur sur un objet YServo accessible en ligne, ou null lorsque l'énumération est terminée.

**servo→registerValueCallback()  
servo.registerValueCallback()****YServo**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**servo→set\_enabled()**

**YServo**

**servo→setEnabled()servo.set\_enabled()**

---

Démarre ou arrête le \$FUNCTION\$.

```
function set_enabled( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_ENABLED\_FALSE, soit Y\_ENABLED\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_enabledAtPowerOn()**  
**servo→setEnabledAtPowerOn()**  
**servo.set\_enabledAtPowerOn()**

YServo

Configure l'état du générateur de signal de commande du servo au démarrage du module.

```
function set_enabledAtPowerOn( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** soit Y\_ENABLEDATPOWERON\_FALSE, soit Y\_ENABLEDATPOWERON\_TRUE

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**servo→set\_logicalName()** YServo  
**servo→setLogicalName()servo.set\_logicalName()**

Modifie le nom logique du servo.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du servo.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_neutral()****YServo****servo→setNeutral()servo.set\_neutral()**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
function set_neutral( ByVal newval As Integer) As Integer
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

**Paramètres :**

**newval** un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_position()**

**YServo**

**servo→setPosition()servo.set\_position()**

---

Modifie immédiatement la consigne de position du servo.

```
function set_position( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant immédiatement la consigne de position du servo

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_positionAtPowerOn()**  
**servo→setPositionAtPowerOn()**  
**servo.set\_positionAtPowerOn()**

YServo

Configure la position du servo au démarrage du module.

```
function set_positionAtPowerOn( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** un entier

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**servo→set\_range()****YServo****servo→setRange()servo.set\_range()**

Modifie la plage d'utilisation du servo, en pourcents.

```
function set_range( ByVal newval As Integer) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.42. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_temperature.js'></script>
nodejs var yoctolib = require('yoctolib');
var YTemperature = yoctolib.YTemperature;
php require_once('yocto_temperature.php');
cpp #include "yocto_temperature.h"
m #import "yocto_temperature.h"
pas uses yocto_temperature;
vb yocto_temperature.vb
cs yocto_temperature.cs
java import com.yoctopuce.YoctoAPI.YTemperature;
py from yocto_temperature import *

```

### Fonction globales

#### yFindTemperature(func)

Permet de retrouver un capteur de température d'après un identifiant donné.

#### yFirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

### Méthodes des objets YTemperature

#### temperature→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### temperature→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### temperature→get\_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

#### temperature→get\_currentRawValue()

Retourne la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.

#### temperature→get\_currentValue()

Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

#### 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.

**temperature→get\_highestValue()**

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

**temperature→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**temperature→get\_logicalName()**

Retourne le nom logique du capteur de température.

**temperature→get\_lowestValue()**

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

**temperature→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**temperature→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**temperature→get\_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.

**temperature→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**temperature→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**temperature→get\_sensorType()**

Retourne le type de capteur de température utilisé par le module

**temperature→get\_unit()**

Retourne l'unité dans laquelle la température est exprimée.

**temperature→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**temperature→isOnline()**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

**temperature→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

**temperature→load(msValidity)**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

**temperature→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

**temperature→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

**temperature→nextTemperature()**

Continue l'énumération des capteurs de température commencée à l'aide de yFirstTemperature( ).

**temperature→registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

**temperature→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

### 3. Reference

**temperature→set\_highestValue(newval)**

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)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**temperature→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YTemperature.FindTemperature() yFindTemperature()yFindTemperature()

## YTemperature

Permet de retrouver un capteur de température d'après un identifiant donné.

```
function yFindTemperature( ByVal func As String) As YTemperature
```

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()yFirstTemperature()**

---

**YTemperature**

Commence l'énumération des capteurs de température accessibles par la librairie.

```
function yFirstTemperature( ) As YTemperature
```

Utiliser la fonction `YTemperature.nextTemperature( )` pour itérer sur les autres capteurs de température.

**Retourne :**

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

**temperature→calibrateFromPoints()**  
**temperature.calibrateFromPoints()****YTemperature**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

**procedure calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→describe()temperature.describe()****YTemperature**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe() As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de température (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**temperature→get\_advertisedValue()**

**YTemperature**

**temperature→advertisedValue()**

**temperature.get\_advertisedValue()**

---

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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), en degrés Celsius, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule

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, en degrés Celsius, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**temperature→getErrorMessage()**  
**temperature→errorMessage()**  
**temperature.getErrorMessage()**

**YTemperature**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

**function getErrorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

**temperature→get\_errorType()**  
**temperature→errorType()**  
**temperature.get\_errorType()****YTemperature**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**temperature→functionDescriptor()**  
**temperature.get\_functionDescriptor()**

---

**YTemperature**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**temperature→get\_functionId()**  
**temperature→functionId()**  
**temperature.get\_functionId()**

**YTemperature**

Retourne l'identifiant matériel du capteur de température, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de température (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**temperature→get\_hardwareId()**  
**temperature→hardwareId()**  
**temperature.get\_hardwareId()**

**YTemperature**

Retourne l'identifiant matériel unique du capteur de température au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de température (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**temperature→get\_highestValue()**  
**temperature→highestValue()**  
**temperature.get\_highestValue()**

**YTemperature**

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**temperature→get\_logFrequency()**  
**temperature→logFrequency()**  
**temperature.get\_logFrequency()**

**YTemperature**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**temperature→get\_logicalName()**  
**temperature→logicalName()**  
**temperature.get\_logicalName()**

**YTemperature**

Retourne le nom logique du capteur de température.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de température.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**temperature→get\_lowestValue()**  
**temperature→lowestValue()**  
**temperature.get\_lowestValue()**

**YTemperature**

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

**function get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_LOWESTVALUE\_INVALID.

**temperature→get\_module()****YTemperature****temperature→module()temperature.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**temperature→get\_recordedData()**  
**temperature→recordedData()**  
**temperature.get\_recordedData()**

**YTemperature**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

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.

function **get\_reportFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

**temperature→get\_resolution()**  
**temperature→resolution()**  
**temperature.get\_resolution()**

---

**YTemperature**

Retourne la résolution des valeurs mesurées.

**function get\_resolution( ) As Double**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**temperature→get\_sensorType()**  
**temperature→sensorType()**  
**temperature.get\_sensorType()**

**YTemperature**

Retourne le type de capteur de température utilisé par le module

```
function get_sensorType( ) As Integer
```

**Retourne :**

une valeur parmi Y\_SENSORTYPE\_DIGITAL, Y\_SENSORTYPE\_TYPE\_K,  
Y\_SENSORTYPE\_TYPE\_E, Y\_SENSORTYPE\_TYPE\_J, Y\_SENSORTYPE\_TYPE\_N,  
Y\_SENSORTYPE\_TYPE\_R, Y\_SENSORTYPE\_TYPE\_S, Y\_SENSORTYPE\_TYPE\_T,  
Y\_SENSORTYPE\_PT100\_4WIRES, Y\_SENSORTYPE\_PT100\_3WIRES et  
Y\_SENSORTYPE\_PT100\_2WIRES représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y\_SENSORTYPE\_INVALID.

**temperature→get\_unit()**

**YTemperature**

**temperature→unit()temperature.get\_unit()**

---

Retourne l'unité dans laquelle la température est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**temperature→get(userData)****YTemperature****temperature→userData()temperature.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

## temperature→isOnline()temperature.isOnline()

## YTemperature

---

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de température est joignable, false sinon

**temperature→load()temperature.load()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

**procedure loadCalibrationPoints( )**

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→nextTemperature()  
temperature.nextTemperature()****YTemperature**

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

```
function nextTemperature( ) As YTemperature
```

**Retourne :**

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**temperature→registerTimedReportCallback()  
temperature.registerTimedReportCallback()****YTemperature**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

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.

**temperature→registerValueCallback()  
temperature.registerValueCallback()****YTemperature**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**temperature→set\_highestValue()**  
**temperature→setHighestValue()**  
**temperature.set\_highestValue()**

**YTemperature**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logFrequency()**  
**temperature→setLogFrequency()**  
**temperature.set\_logFrequency()**

**YTemperature**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**function set\_logFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_logicalName()**  
**temperature→setLogicalName()**  
**temperature.set\_logicalName()**

**YTemperature**

Modifie le nom logique du capteur de température.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de température.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_lowestValue()**  
**temperature→setLowestValue()**  
**temperature.set\_lowestValue()**

YTemperature

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_reportFrequency()**  
**temperature→setReportFrequency()**  
**temperature.set\_reportFrequency()**

**YTemperature**

---

Modifie la fréquence de notification périodique des valeurs mesurées.

**function set\_reportFrequency( ByVal newval As String) As Integer**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_resolution()**  
**temperature→setResolution()**  
**temperature.set\_resolution()**

**YTemperature**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**temperature→set\_sensorType()**  
**temperature→setSensorType()**  
**temperature.set\_sensorType()**

**YTemperature**

Change le type de senseur utilisé par le module.

```
function set_sensorType( ByVal newval As Integer ) As Integer
```

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).

procedure **set(userData)** ByVal **data** As Object

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.43. Interface de la fonction Tilt

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_tilt.js'></script>
nodejs var yoctolib = require('yoctolib');
var YTilt = yoctolib.YTilt;
php require_once('yocto_tilt.php');
cpp #include "yocto_tilt.h"
m #import "yocto_tilt.h"
pas uses yocto_tilt;
vb yocto_tilt.vb
cs yocto_tilt.cs
java import com.yoctopuce.YoctoAPI.YTilt;
py from yocto_tilt import *

```

### Fonction globales

#### yFindTilt(func)

Permet de retrouver un inclinomètre d'après un identifiant donné.

#### yFirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

### Méthodes des objets YTilt

#### tilt→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### tilt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format TYPE (NAME) = SERIAL . FUNCTIONID.

#### tilt→get\_advertisedValue()

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

#### tilt→get\_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

#### tilt→get\_currentValue()

Retourne la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule.

#### 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>

### 3. Reference

---

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

**tilt→set\_logicalName(newval)**

Modifie le nom logique de l'inclinomètre.

**tilt→set\_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

**tilt→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**tilt→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**tilt→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**tilt→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YTilt.FindTilt()****YTilt****yFindTilt()yFindTilt()**

Permet de retrouver un inclinomètre d'après un identifiant donné.

```
function yFindTilt( ByVal func As String) As YTilt
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'inclinomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTilt.isOnLine()` pour tester si l'inclinomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'inclinomètre sans ambiguïté

**Retourne :**

un objet de classe `YTilt` qui permet ensuite de contrôler l'inclinomètre.

## YTilt.FirstTilt() yFirstTilt()yFirstTilt()

YTilt

Commence l'énumération des inclinomètres accessibles par la librairie.

```
function yFirstTilt( ) As YTilt
```

Utiliser la fonction YTilt.nextTilt( ) pour itérer sur les autres inclinomètres.

**Retourne :**

un pointeur sur un objet YTilt, correspondant au premier inclinomètre accessible en ligne, ou null si il n'y a pas de inclinomètres disponibles.

**tilt→calibrateFromPoints()|tilt.calibrateFromPoints()****YTilt**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

procedure **calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→describe()tilt.describe()****YTilt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe() As String
```

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).

```
function get_advertisedValue( ) As String
```

**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.

---

<b>tilt→get_currentRawValue()</b>	<b>YTilt</b>
<b>tilt→currentRawValue()tilt.get_currentRawValue()</b>	

---

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

```
function get_currentRawValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule

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, en degrés, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

<b>tilt→getErrorMessage()</b>	<b>YTilt</b>
<b>tilt→errorMessage()tilt.getErrorMessage()</b>	

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
function getErrorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

**tilt→get\_errorType()****YTilt****tilt→errorType()tilt.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
function get_errorType( ) As YRETCODE
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

---

<b>tilt→get_functionDescriptor()</b>	<b>YTilt</b>
<b>tilt→functionDescriptor()tilt.get_functionDescriptor()</b>	

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**tilt→get\_functionId()**

YTilt

**tilt→functionId()tilt.get\_functionId()**

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

<b>tilt→getHardwareId()</b>	<b>YTilt</b>
<b>tilt→hardwareId()tilt.getHardwareId()</b>	

Retourne l'identifiant matériel unique de l'inclinomètre au format SERIAL.FUNCTIONID.

```
function getHardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'inclinomètre (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'inclinomètre (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**tilt→get\_highestValue()****YTilt****tilt→highestValue()tilt.get\_highestValue()**

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

```
function get_highestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**tilt→get\_logFrequency()**

**YTilt**

**tilt→logFrequency()tilt.get\_logFrequency()**

---

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**function get\_logFrequency( ) As String**

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**tilt→get\_logicalName()****YTilt****tilt→logicalName()tilt.get\_logicalName()**

Retourne le nom logique de l'inclinomètre.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'inclinomètre.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

<b>tilt→get_lowestValue()</b>	<b>YTilt</b>
<b>tilt→lowestValue()tilt.get_lowestValue()</b>	

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

```
function get_lowestValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

**tilt→get\_module()****YTilt****tilt→module()tilt.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

<b>tilt→get_recordedData()</b>	<b>YTilt</b>
<b>tilt→recordedData()tilt.get_recordedData()</b>	

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**tilt→get\_reportFrequency()**

YTilt

**tilt→reportFrequency()tilt.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( ) As String
```

**Retourne :**

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y\_REPORTFREQUENCY\_INVALID.

<b>tilt→get_resolution()</b>	<b>YTilt</b>
<b>tilt→resolution()tilt.get_resolution()</b>	

---

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**tilt→get\_unit()****YTilt****tilt→unit()tilt.get\_unit()**

Retourne l'unité dans laquelle l'inclinaison est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle l'inclinaison est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**tilt→get(userData)**

YTilt

**tilt→userData()tilt.get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData)( ) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**tilt→isOnline()tilt.isOnline()****YTilt**

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache de l'inclinomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

`true` si l'inclinomètre est joignable, `false` sinon

**tilt→load()tilt.load()****YTilt**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

```
procedure loadCalibrationPoints( )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **tilt→nextTilt()tilt.nextTilt()**

**YTilt**

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

```
function nextTilt( ) As YTilt
```

**Retourne :**

un pointeur sur un objet YTilt accessible en ligne, ou null lorsque l'énumération est terminée.

**tilt→registerTimedReportCallback()  
tilt.registerTimedReportCallback()**

YTilt

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**tilt→registerValueCallback()  
tilt.registerValueCallback()****YTilt**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**tilt→set\_highestValue()****YTilt****tilt→setHighestValue()tilt.set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**tilt→set\_logFrequency()** YTilt  
**tilt→setLogFrequency()tilt.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_logicalName()**

YTilt

**tilt→setLogicalName()tilt.set\_logicalName()**

Modifie le nom logique de l'inclinomètre.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique de l'inclinomètre.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_lowestValue()** YTilt  
**tilt→setLowestValue()tilt.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**tilt→set\_reportFrequency()**

YTilt

**tilt→setReportFrequency()tilt.set\_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>tilt→set_resolution()</b>	<b>YTilt</b>
<b>tilt→setResolution()tilt.set_resolution()</b>	

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.44. Interface de la fonction Voc

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_voc.js'></script>
nodejs	var yoctolib = require('yoctolib');
	var YVoc = yoctolib.YVoc;
php	require_once('yocto_voc.php');
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 renvoyée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule.

#### voc→get\_currentValue()

Retourne la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule.

#### 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.

**voc→get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.

**voc→get\_highestValue()**

Retourne la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module.

**voc→get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

**voc→get\_logicalName()**

Retourne le nom logique du capteur de Composés Organiques Volatils.

**voc→get\_lowestValue()**

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

**voc→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**voc→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**voc→get\_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**voc→get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

**voc→get\_resolution()**

Retourne la résolution des valeurs mesurées.

**voc→get\_unit()**

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

**voc→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**voc→isOnline()**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

**voc→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

**voc→load(msValidity)**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

**voc→loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

**voc→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

**voc→nextVoc()**

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de yFirstVoc( ).

**voc→registerTimedReportCallback(callback)**

### 3. Reference

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.

#### **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.

#### **voc→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

#### **voc→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

#### **voc→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

#### **voc→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YVoc.FindVoc() yFindVoc()yFindVoc()

YVoc

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

```
function yFindVoc( ByVal func As String) As YVoc
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de Composés Organiques Volatils soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoc.isOnLine()` pour tester si le capteur de Composés Organiques Volatils est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le capteur de Composés Organiques Volatils sans ambiguïté

### Retourne :

un objet de classe `YVoc` qui permet ensuite de contrôler le capteur de Composés Organiques Volatils.

## YVoc.FirstVoc() yFirstVoc()yFirstVoc()

**YVoc**

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

```
function yFirstVoc( ) As YVoc
```

Utiliser la fonction `YVoc.nextVoc()` pour itérer sur les autres capteurs de Composés Organiques Volatils.

**Retourne :**

un pointeur sur un objet `YVoc`, correspondant au premier capteur de Composés Organiques Volatils accessible en ligne, ou `null` si il n'y a pas de capteurs de Composés Organiques Volatils disponibles.

**voc→calibrateFromPoints()voc.calibrateFromPoints()****YVoc**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

procedure **calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→describe()voc.describe()****YVoc**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format TYPE (NAME )=SERIAL .FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**voc→get\_advertisedValue()****YVoc****voc→advertisedValue()voc.get\_advertisedValue()**

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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 renvoyée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule.

```
function get_currentRawValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

**voc→get\_currentValue()****YVoc****voc→currentValue()voc.get\_currentValue()**

Retourne la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule.

function **get\_currentValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne **Y\_CURRENTVALUE\_INVALID**.

**voc→getErrorMessage()**

**YVoc**

**voc→errorMessage()voc.getErrorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

**function getErrorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

**voc→get\_errorType()****YVoc****voc→errorType()voc.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**voc->functionDescriptor()**  
**voc.get\_functionDescriptor()**

**YVoc**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voc→get\_functionId()****YVoc****voc→functionId()voc.get\_functionId()**

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voc→get\_hardwareId()****YVoc****voc→hardwareId()voc.get\_hardwareId()**

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

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 depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_HIGHESTVALUE\_INVALID**.

**voc→get\_logFrequency()** YVoc  
**voc→logFrequency()voc.get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( ) As String
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

**voc→get\_logicalName()****YVoc****voc→logicalName()voc.get\_logicalName()**

Retourne le nom logique du capteur de Composés Organiques Volatils.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**voc→get\_lowestValue()**

**YVoc**

**voc→lowestValue()voc.get\_lowestValue()**

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

function **get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_LOWESTVALUE\_INVALID**.

**voc→get\_module()****YVoc****voc→module()voc.get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retornée ne sera pas joignable.

**Retourne :**

une instance de YModule

**voc→get\_recordedData()** YVoc  
**voc→recordedData()voc.get\_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voc→get\_reportFrequency()****YVoc****voc→reportFrequency()voc.get\_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( ) As String
```

**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()**  
**voc→resolution()voc.get\_resolution()**

---

YVoc

Retourne la résolution des valeurs mesurées.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**voc→get\_unit()****YVoc****voc→unit()voc.get\_unit()**

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle le taux de VOC estimé est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**voc→get(userData)**

YVoc

**voc→userData()voc.get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**voc→isOnline()voc.isOnline()****YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du capteur de Composés Organiques Volatils sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de Composés Organiques Volatils est joignable, false sinon

**voc→load()voc.load()****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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()****YVoc****voc.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

```
procedure loadCalibrationPoints( )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## voc→nextVoc()voc.nextVoc()

YVoc

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

```
function nextVoc( ) As YVoc
```

**Retourne :**

un pointeur sur un objet YVoc accessible en ligne, ou `null` lorsque l'énumération est terminée.

**voc→registerTimedReportCallback()**  
**voc.registerTimedReportCallback()****YVoc**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback( ) As Integer**

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.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**voc→set\_highestValue()****YVoc****voc→setHighestValue()voc.set\_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**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.

**voc→set\_logFrequency()** YVoc  
**voc→setLogFrequency()voc.set\_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set\_logicalName()****YVoc****voc→setLogicalName()voc.set\_logicalName()**

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
function set_logicalName( ByVal newval As String) As Integer
```

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.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**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.

**voc→set\_reportFrequency()**  
**voc→setReportFrequency()**  
**voc.set\_reportFrequency()**

YVoc

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

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()  
voc→setResolution()voc.set\_resolution()****YVoc**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

**Paramètres :**

**newval** une valeur numérique représentant la résolution des valeurs physique mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voc→set(userData)****YVoc****voc→setUserData()|voc.set(userData)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.45. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_voltage.js'></script>
nodejs var yoctolib = require('yoctolib');
var YVoltage = yoctolib.YVoltage;
php require_once('yocto_voltage.php');
cpp #include "yocto_voltage.h"
m #import "yocto_voltage.h"
pas uses yocto_voltage;
vb yocto_voltage.vb
cs yocto_voltage.cs
java import com.yoctopuce.YoctoAPI.YVoltage;
py from yocto_voltage import *

```

### Fonction globales

#### **yFindVoltage(func)**

Permet de retrouver un capteur de tension d'après un identifiant donné.

#### **yFirstVoltage()**

Commence l'énumération des capteurs de tension accessibles par la librairie.

### Méthodes des objets YVoltage

#### **voltage→calibrateFromPoints(rawValues, refValues)**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

#### **voltage→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### **voltage→get\_advertisedValue()**

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

#### **voltage→get\_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

#### **voltage→get\_currentValue()**

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

#### **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 depuis le démarrage du module.
<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 depuis le démarrage du module.
<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.
<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.

**voltage→set\_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

**voltage→set\_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

**voltage→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**voltage→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YVoltage.FindVoltage()

## YVoltage

### yFindVoltage()yFindVoltage()

Permet de retrouver un capteur de tension d'après un identifiant donné.

```
function yFindVoltage( ByVal func As String) As YVoltage
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltage.isOnLine()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

#### Paramètres :

**func** une chaîne de caractères qui référence le capteur de tension sans ambiguïté

#### Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

## **YVoltage.FirstVoltage() yFirstVoltage()yFirstVoltage()**

**YVoltage**

Commence l'énumération des capteurs de tension accessibles par la librairie.

```
function yFirstVoltage( ) As YVoltage
```

Utiliser la fonction `YVoltage.nextVoltage( )` pour itérer sur les autres capteurs de tension.

**Retourne :**

un pointeur sur un objet `YVoltage`, correspondant au premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

**voltage→calibrateFromPoints()**  
**voltage.calibrateFromPoints()****YVoltage**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

procedure **calibrateFromPoints( )**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

**Paramètres :**

**rawValues** tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

**refValues** tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→describe()voltage.describe()****YVoltage**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format TYPE (NAME )=SERIAL.FUNCTIONID.

```
function describe() As String
```

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 tension (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)
```

**voltage→get\_advertisedValue()**  
**voltage→advertisedValue()**  
**voltage.get\_advertisedValue()****YVoltage**

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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), en Volt, sous forme de nombre à virgule.

function **get\_currentRawValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur brute renvoyée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTRAWVALUE\_INVALID.

---

<b>voltage→get_currentValue()</b>	<b>YVoltage</b>
<b>voltage→currentValue()voltage.get_currentValue()</b>	

---

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

```
function get_currentValue( ) As Double
```

**Retourne :**

une valeur numérique représentant la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y\_CURRENTVALUE\_INVALID.

**voltage→get\_errorMessage()**

**YVoltage**

**voltage→errorMessage()voltage.get\_errorMessage()**

---

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function get_errorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

**voltage→get\_errorType()****YVoltage****voltage→errorType()voltage.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**voltage→functionDescriptor()**  
**voltage.get\_functionDescriptor()**

**YVoltage**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**voltage→get\_functionId()****YVoltage****voltage→functionId()voltage.get\_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**voltage→get\_hardwareId()**

**YVoltage**

**voltage→hardwareId()voltage.get\_hardwareId()**

---

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.

function **get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de tension (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le capteur de tension (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

---

<b>voltage→get_highestValue()</b>	<b>YVoltage</b>
<b>voltage→highestValue()voltage.get_highestValue()</b>	

---

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

function **get\_highestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur maximale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y\_HIGHESTVALUE\_INVALID.

**voltage→get\_logFrequency()**

**YVoltage**

**voltage→logFrequency()voltage.get\_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( ) As String
```

**Retourne :**

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y\_LOGFREQUENCY\_INVALID.

---

<b>voltage→get_logicalName()</b>	<b>YVoltage</b>
<b>voltage→logicalName()voltage.get_logicalName()</b>	

---

Retourne le nom logique du capteur de tension.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du capteur de tension.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**voltage→get\_lowestValue()**

**YVoltage**

**voltage→lowestValue()voltage.get\_lowestValue()**

---

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

function **get\_lowestValue( ) As Double**

**Retourne :**

une valeur numérique représentant la valeur minimale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne **Y\_LOWESTVALUE\_INVALID**.

**voltage→get\_module()****YVoltage****voltage→module()voltage.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

---

<b>voltage→get_recordedData()</b>	<b>YVoltage</b>
<b>voltage→recordedData()voltage.get_recordedData()</b>	

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

**function get\_recordedData( ) As YDataSet**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

**Paramètres :**

**startTime** le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

**endTime** la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

**Retourne :**

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

**voltage→get\_reportFrequency()**  
**voltage→reportFrequency()**  
**voltage.get\_reportFrequency()**

**YVoltage**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( ) As String
```

**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.

```
function get_resolution( ) As Double
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

**Retourne :**

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y\_RESOLUTION\_INVALID.

**voltage→get\_unit()****YVoltage****voltage→unit()voltage.get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
function get_unit( ) As String
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**voltage→get(userData)**

**YVoltage**

**voltage→userData()voltage.get(userData())**

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**voltage→isOnline()voltage.isOnline()****YVoltage**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le capteur de tension est joignable, false sinon

**voltage→load()voltage.load()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être 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.

```
procedure loadCalibrationPoints( )
```

**Paramètres :**

**rawValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

**refValues** tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## voltage→nextVoltage()voltage.nextVoltage()

YVoltage

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

```
function nextVoltage( ) As YVoltage
```

**Retourne :**

un pointeur sur un objet YVoltage accessible en ligne, ou null lorsque l'énumération est terminée.

**voltage→registerTimedReportCallback()**  
**voltage.registerTimedReportCallback()****YVoltage**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

**voltage→registerValueCallback()  
voltage.registerValueCallback()****YVoltage**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

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.

```
function set_highestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur maximale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

<b>voltage→set_logFrequency()</b>	<b>YVoltage</b>
<b>voltage→setLogFrequency()</b>	
<b>voltage.set_logFrequency()</b>	

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

**voltage→set\_logicalName()** YVoltage  
**voltage→setLogicalName()voltage.set\_logicalName()**

---

Modifie le nom logique du capteur de tension.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du capteur de tension.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_lowestValue()**

**YVoltage**

**voltage→setLowestValue()voltage.set\_lowestValue()**

---

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( ByVal newval As Double) As Integer
```

**Paramètres :**

**newval** une valeur numérique représentant la mémoire de valeur minimale observée

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**voltage→set\_reportFrequency()**  
**voltage→setReportFrequency()**  
**voltage.set\_reportFrequency()**

YVoltage

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( ByVal newval As String) As Integer
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

**Paramètres :**

**newval** une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

<b>voltage→set_resolution()</b>	<b>YVoltage</b>
<b>voltage→setResolution()voltage.set_resolution()</b>	

---

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( ByVal newval As Double) As Integer
```

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.

**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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.46. Interface de la fonction Source de tension

La librairie de programmation Yoctopuce permet de commander la tension de sortie du module. Vous pouvez affecter une valeur fixe, ou faire des transitions de voltage.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_vsource.js'></script>
php	require_once('yocto_vsource.php');
cpp	#include "yocto_vsource.h"
m	#import "yocto_vsource.h"
pas	uses yocto_vsource;
vb	yocto_vsource.vb
cs	yocto_vsource.cs
java	import com.yoctopuce.YoctoAPI.YVSource;
py	from yocto_vsource import *

<b>Fonction globales</b>	
<b>yFindVSource(func)</b>	Permet de retrouver une source de tension d'après un identifiant donné.
<b>yFirstVSource()</b>	Commence l'énumération des sources de tension accessibles par la librairie.
<b>Méthodes des objets YVSource</b>	
<b>vsource→describe()</b>	Retourne un court texte décrivant la fonction au format TYPE ( NAME ) = SERIAL . FUNCTIONID.
<b>vsource→get_advertisedValue()</b>	Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
<b>vsource→get_errorMessage()</b>	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>vsource→get_errorType()</b>	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
<b>vsource→get_extPowerFailure()</b>	Rend TRUE si le voltage de l'alimentation externe est trop bas.
<b>vsource→get_failure()</b>	Indique si le module est en condition d'erreur.
<b>vsource→get_friendlyName()</b>	Retourne un identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.
<b>vsource→get_functionDescriptor()</b>	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
<b>vsource→get_functionId()</b>	Retourne l'identifiant matériel de la fonction, sans référence au module.
<b>vsource→get_hardwareId()</b>	Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.
<b>vsource→get_logicalName()</b>	Retourne le nom logique de la source de tension.
<b>vsource→get_module()</b>	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>vsource→get_module_async(callback, context)</b>	

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**vsouce→get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

**vsouce→get\_overHeat()**

Rend TRUE si le module est en surchauffe.

**vsouce→get\_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

**vsouce→get\_regulationFailure()**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

**vsouce→get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

**vsouce→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

**vsouce→get\_voltage()**

Retourne la valeur de la commande de tension de sortie en mV

**vsouce→isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

**vsouce→isOnline\_async(callback, context)**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

**vsouce→load(msValidity)**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**vsouce→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

**vsouce→nextVSource()**

Continue l'énumération des sources de tension commencée à l'aide de yFirstVSource( ).

**vsouce→pulse(voltage, ms\_duration)**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

**vsouce→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**vsouce→set\_logicalName(newval)**

Modifie le nom logique de la source de tension.

**vsouce→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**vsouce→set\_voltage(newval)**

Règle la tension de sortie du module (en millivolts).

**vsouce→voltageMove(target, ms\_duration)**

Déclenche une variation constante de la sortie vers une valeur donnée.

**vsouce→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## yFindVSource() — YVSource.FindVSource()yFindVSource()

Permet de retrouver une source de tension d'après un identifiant donné.

**function yFindVSource( ByVal func As String) As YVSource**

## yFindVSource() — YVSource.FindVSource()yFindVSource()

Permet de retrouver une source de tension d'après un identifiant donné.

```

js function yFindVSource( func)
php function yFindVSource( $func)
cpp YVSource* yFindVSource( const string& func)
m YVSource* yFindVSource( NSString* func)
pas function yFindVSource( func: string): TYVSource
vb function yFindVSource( ByVal func As String) As YVSource
cs YVSource FindVSource( string func)
java YVSource FindVSource( String func)
py def FindVSource( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

**func** une chaîne de caractères qui référence la source de tension sans ambiguïté

### Retourne :

un objet de classe `YVSource` qui permet ensuite de contrôler la source de tension.

**yFirstVSource() —****YVSource****YVSource.FirstVSource()yFirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

```
function yFirstVSource( ) As YVSource
```

**yFirstVSource() — YVSource.FirstVSource()yFirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

```
js   function yFirstVSource( )
php  function yFirstVSource( )
cpp  YVSource* yFirstVSource( )
m    YVSource* yFirstVSource( )
pas   function yFirstVSource( ):TYVSource
vb    function yFirstVSource( ) As YVSource
cs    YVSource FirstVSource( )
java  YVSource FirstVSource( )
py    def FirstVSource( )
```

Utiliser la fonction `YVSource.nextVSource()` pour itérer sur les autres sources de tension.

**Retourne :**

un pointeur sur un objet `YVSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de sources de tension disponibles.

**vsouce→describe()vsouce.describe()****YVSource**

Retourne un court texte décrivant la fonction au format TYPE ( NAME )=SERIAL . FUNCTIONID.

function **describe( )** As String

**vsouce→describe()vsouce.describe()**

Retourne un court texte décrivant la fonction au format TYPE ( NAME )=SERIAL . FUNCTIONID.

js	function <b>describe( )</b>
php	function <b>describe( )</b>
cpp	string <b>describe( )</b>
m	- <b>(NSString*) describe</b>
pas	function <b>describe( )</b> : string
vb	function <b>describe( )</b> As String
cs	string <b>describe( )</b>
java	String <b>describe( )</b>

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant la fonction (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**vsource→get\_advertisedValue()**  
**vsource→advertisedValue()**  
**vsource.get\_advertisedValue()**

**YVSource**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

function **get\_advertisedValue( ) As String**

**vsource→get\_advertisedValue()**  
**vsource→advertisedValue()vsource.get\_advertisedValue()**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

**js** function **get\_advertisedValue( )**  
**php** function **get\_advertisedValue( )**  
**cpp** string **get\_advertisedValue( )**  
**m** -(NSString\*) advertisedValue  
**pas** function **get\_advertisedValue( ): string**  
**vb** function **get\_advertisedValue( ) As String**  
**cs** string **get\_advertisedValue( )**  
**java** String **get\_advertisedValue( )**  
**py** def **get\_advertisedValue( )**  
**cmd** YVSource **target get\_advertisedValue**

**Retourne :**

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères)

En cas d'erreur, déclenche une exception ou retourne **Y\_ADVERTISEDVALUE\_INVALID**.

**vsouce→get\_errorMessage()**  
**vsouce→errorMessage()**  
**vsouce.get\_errorMessage()**

**YVSource**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

function **get\_errorMessage( ) As String**

**vsouce→get\_errorMessage()**  
**vsouce→errorMessage()vsouce.get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

**js** function **get\_errorMessage( )**  
**php** function **get\_errorMessage( )**  
**cpp** string **get\_errorMessage( )**  
**m** -(NSString\*) errorMessage  
**pas** function **get\_errorMessage( ): string**  
**vb** function **get\_errorMessage( ) As String**  
**cs** string **get\_errorMessage( )**  
**java** String **get\_errorMessage( )**  
**py** def **get\_errorMessage( )**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

**vsouce→get\_errorType()****YVSource****vsouce→errorType()vsouce.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
function get_errorType( ) As YRETCODE
```

**vsouce→get\_errorType()****vsouce→errorType()vsouce.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

js	function 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

**vsouce→get\_extPowerFailure()**  
**vsouce→extPowerFailure()**  
**vsouce.get\_extPowerFailure()**

YVSource

Rend TRUE si le voltage de l'alimentation externe est trop bas.

function **get\_extPowerFailure( ) As Integer**

**vsouce→get\_extPowerFailure()**  
**vsouce→extPowerFailure()vsouce.get\_extPowerFailure()**

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.

---

**vsource→get\_failure()**  
**vsource→failure()vsource.get\_failure()**


---

Indique si le module est en condition d'erreur.

```
function get_failure( ) As Integer
```

---

**vsource→get\_failure()**  
**vsource→failure()vsource.get\_failure()**


---

Indique si le module est en condition d'erreur.

```
js   function get_failure( )
php  function get_failure( )
cpp  Y_FAILURE_enum get_failure( )
m    -(Y_FAILURE_enum) failure
pas   function get_failure( ): Integer
vb    function get_failure( ) As Integer
cs    int get_failure( )
java  int get_failure( )
py    def get_failure( )
cmd   YVSource target get_failure
```

Il possible de savoir de quelle erreur il s'agit en testant get\_overheat, get\_overcurrent etc... Lorsqu'un condition d'erreur est rencontrée, la tension de sortie est mise à zéro et ne peut pas être changée tant la fonction reset() n'aura pas appellée.

**Retourne :**

soit Y\_FAILURE\_FALSE, soit Y\_FAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_FAILURE\_INVALID.

**YVSource**

**vsource→get\_functionDescriptor()**  
**vsource→functionDescriptor()**  
**vsource.get\_vsourceDescriptor()**

**YVSource**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

function **get\_functionDescriptor( ) As YFUN\_DESCR**

**vsource→get\_functionDescriptor()**  
**vsource→functionDescriptor()vsource.get\_vsourceDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

js function **get\_functionDescriptor( )**  
php function **get\_functionDescriptor( )**  
cpp YFUN\_DESCR **get\_functionDescriptor( )**  
m -(YFUN\_DESCR) **functionDescriptor**  
pas function **get\_functionDescriptor( ): YFUN\_DESCR**  
vb function **get\_functionDescriptor( ) As YFUN\_DESCR**  
cs YFUN\_DESCR **get\_functionDescriptor( )**  
java String **get\_functionDescriptor( )**  
py def **get\_functionDescriptor( )**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**vsouce→get\_functionId()****YVSource****vsouce→functionId()vsouce.get\_vsourceld()**

Retourne l'identifiant matériel de la fonction, sans référence au module.

```
function get_functionId( ) As String
```

**vsouce→get\_functionId()****vsouce→functionId()vsouce.get\_vsourceld()**

Retourne l'identifiant matériel de la fonction, sans référence au module.

```
js   function get_functionId( )
php  function get_functionId( )
cpp  string get_functionId( )
m    -(NSString*)functionId
vb   function get_functionId( ) As String
cs   string get_functionId( )
java String get_functionId( )
```

Par exemple relay1.

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y\_FUNCTIONID\_INVALID.

**vsource→get\_hardwareId()****YVSource****vsource→hardwareId()vsource.get\_hardwareId()**

Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

function **get\_hardwareId( ) As String**

**vsource→get\_hardwareId()****vsource→hardwareId()vsource.get\_hardwareId()**

Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

**js** function **get\_hardwareId( )**

**php** function **get\_hardwareId( )**

**cpp** string **get\_hardwareId( )**

**m** -(NSString\*) hardwareId

**vb** function **get\_hardwareId( ) As String**

**cs** string **get\_hardwareId( )**

**java** String **get\_hardwareId( )**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant la fonction (ex: RELAYL01-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**vsource→get\_logicalName()****YVSource****vsource→logicalName()vsource.get\_logicalName()**

Retourne le nom logique de la source de tension.

```
function get_logicalName( ) As String
```

**vsource→get\_logicalName()****vsource→logicalName()vsource.get\_logicalName()**

Retourne le nom logique de la source de tension.

```
js   function get_logicalName( )
php  function get_logicalName( )
cpp  string get_logicalName( )
m    -(NSString*) logicalName
pas   function get_logicalName( ): string
vb    function get_logicalName( ) As String
cs    string get_logicalName( )
java  String get_logicalName( )
py    def get_logicalName( )
cmd   YVSource target get_logicalName
```

**Retourne :**

une chaîne de caractères représentant le nom logique de la source de tension

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**vsouce→get\_module()**  
**vsouce→module()vsouce.get\_module()**

**YVSource**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get\_module( ) As YModule**

**vsouce→get\_module()**  
**vsouce→module()vsouce.get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**js** function **get\_module( )**  
**php** function **get\_module( )**  
**cpp** YModule \* **get\_module( )**  
**m** -(YModule\*) module  
**pas** function **get\_module( ): TYModule**  
**vb** function **get\_module( ) As YModule**  
**cs** YModule **get\_module( )**  
**java** YModule **get\_module( )**  
**py** def **get\_module( )**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

**vsource→get\_overCurrent()****YVSource****vsource→overCurrent()vsource.get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
function get_overCurrent( ) As Integer
```

**vsource→get\_overCurrent()****vsource→overCurrent()vsource.get\_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
js function 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.

---

<b>vsouce→get_overHeat()</b>	<b>YVSource</b>
<b>vsouce→overHeat() vsouce.get_overHeat()</b>	

---

Rend TRUE si le module est en surchauffe.

```
function get_overHeat( ) As Integer
```

---

<b>vsouce→get_overHeat()</b>
<b>vsouce→overHeat() vsouce.get_overHeat()</b>

---

Rend TRUE si le module est en surchauffe.

```
js   function get_overHeat( )
php  function get_overHeat( )
cpp  Y_OVERHEAT_enum get_overHeat( )
m    -(Y_OVERHEAT_enum) overHeat
pas   function get_overHeat( ): Integer
vb    function get_overHeat( ) As Integer
cs    int get_overHeat( )
java  int get_overHeat( )
py    def get_overHeat( )
cmd   YVSource target get_overHeat
```

**Retourne :**

soit Y\_OVERHEAT\_FALSE, soit Y\_OVERHEAT\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERHEAT\_INVALID.

**vsource→get\_overLoad()****YVSource****vsource→overLoad()vsource.get\_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

```
function get_overLoad( ) As Integer
```

**vsource→get\_overLoad()****vsource→overLoad()vsource.get\_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

```
js function get_overLoad( )
php function get_overLoad( )
cpp Y_OVERLOAD_enum get_overLoad( )
m -(Y_OVERLOAD_enum) overLoad
pas function get_overLoad( ): Integer
vb function get_overLoad( ) As Integer
cs int get_overLoad( )
java int get_overLoad( )
py def get_overLoad( )
cmd YVSource target get_overLoad
```

**Retourne :**

soit Y\_OVERLOAD\_FALSE, soit Y\_OVERLOAD\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_OVERLOAD\_INVALID.

---

<b>vsOURCE→get_regulationFailure()</b>	<b>YVSource</b>
<b>vsOURCE→regulationFailure()</b>	
<b>vsOURCE.get_regulationFailure()</b>	

---

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

function **get\_regulationFailure( ) As Integer**

---

<b>vsOURCE→get_regulationFailure()</b>
<b>vsOURCE→regulationFailure()vsOURCE.get_regulationFailure()</b>

---

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

---

<b>js</b>	function <b>get_regulationFailure( )</b>
<b>php</b>	function <b>get_regulationFailure( )</b>
<b>cpp</b>	Y_REGULATIONFAILURE_enum <b>get_regulationFailure( )</b>
<b>m</b>	-{Y_REGULATIONFAILURE_enum} regulationFailure
<b>pas</b>	function <b>get_regulationFailure( ): Integer</b>
<b>vb</b>	function <b>get_regulationFailure( ) As Integer</b>
<b>cs</b>	int <b>get_regulationFailure( )</b>
<b>java</b>	int <b>get_regulationFailure( )</b>
<b>py</b>	def <b>get_regulationFailure( )</b>
<b>cmd</b>	YVSource target <b>get_regulationFailure</b>

---

**Retourne :**

soit Y\_REGULATIONFAILURE\_FALSE, soit Y\_REGULATIONFAILURE\_TRUE

En cas d'erreur, déclenche une exception ou retourne Y\_REGULATIONFAILURE\_INVALID.

**vsource→get\_unit()****YVSource****vsource→unit()vsource.get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
function get_unit( ) As String
```

**vsource→get\_unit()****vsource→unit()vsource.get\_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
js   function get_unit( )
php  function get_unit( )
cpp  string get_unit( )
m    -(NSString*) unit
pas  function get_unit( ): string
vb   function get_unit( ) As String
cs   string get_unit( )
java String get_unit( )
py   def get_unit( )
cmd  YVSource target get_unit
```

**Retourne :**

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y\_UNIT\_INVALID.

**vsource→get(userData)**  
**vsource→userData(vsource.get(userData))****YVSource**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

**vsource→get(userData)**  
**vsource→userData(vsource.get(userData))**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

<b>js</b>	function get(userData)
<b>php</b>	function get(userData)
<b>cpp</b>	void * get(userData)
<b>m</b>	-(void*) userData
<b>pas</b>	function get(userData): Tobject
<b>vb</b>	function get(userData) As Object
<b>cs</b>	object get(userData)
<b>java</b>	Object get(userData)
<b>py</b>	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.

**vsource→get\_voltage()****YVSource****vsource→voltage()vsource.get\_voltage()**

Retourne la valeur de la commande de tension de sortie en mV

```
function get_voltage( ) As Integer
```

**vsource→get\_voltage()****vsource→voltage()vsource.get\_voltage()**

Retourne la valeur de la commande de tension de sortie en mV

```
js function get_voltage( )
```

```
php function get_voltage( )
```

```
cpp int get_voltage( )
```

```
m -(int) voltage
```

```
pas function get_voltage( ): LongInt
```

```
vb function get_voltage( ) As Integer
```

```
cs int get_voltage( )
```

```
java int get_voltage( )
```

```
py def get_voltage( )
```

**Retourne :**

un entier représentant la valeur de la commande de tension de sortie en mV

En cas d'erreur, déclenche une exception ou retourne Y\_VOLTAGE\_INVALID.

**vsource→isOnline()vsource.isOnline()****YVSource**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

function **isOnline()** As Boolean

**vsource→isOnline()vsource.isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

js	function <b>isOnline()</b>
php	function <b>isOnline()</b>
cpp	bool <b>isOnline()</b>
m	- <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 fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si la fonction est joignable, false sinon

**vsource→load()vsource.load()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

**vsource→load()vsource.load()**

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.

**vsource→nextVSource()vsource.nextVSource()****YVSource**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

function **nextVSource()** As YVSource

**vsource→nextVSource()vsource.nextVSource()**

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	-( <b>YVSource*</b> ) <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.

**vsourcē→pulse()****YVSource**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
function pulse( ByVal voltage As Integer,
                ByVal ms_duration As Integer) As Integer
```

**vsourcē→pulse()**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

js	function pulse( <b>voltage</b> , <b>ms_duration</b> )
php	function pulse( \$voltage, \$ms_duration)
cpp	int pulse( int <b>voltage</b> , int <b>ms_duration</b> )
m	-( <b>int</b> ) pulse : ( <b>int</b> ) <b>voltage</b> : ( <b>int</b> ) <b>ms_duration</b>
pas	function pulse( <b>voltage</b> : integer, <b>ms_duration</b> : integer): integer
vb	function pulse( ByVal <b>voltage</b> As Integer,                             ByVal <b>ms_duration</b> As Integer) As Integer
cs	int pulse( int <b>voltage</b> , int <b>ms_duration</b> )
java	int pulse( int <b>voltage</b> , int <b>ms_duration</b> )
py	def pulse( <b>voltage</b> , <b>ms_duration</b> )
cmd	YVSource target pulse <b>voltage</b> <b>ms_duration</b>

**Paramètres :**

**voltage** tension demandée, en millivolts

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsource→registerValueCallback()  
vsource.registerValueCallback()****YVSource**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

procedure **registerValueCallback( ByVal callback As GenericUpdateCallback)**

**vsource→registerValueCallback()vsource.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js function registerValueCallback( callback)
php function registerValueCallback( $callback)
cpp void registerValueCallback( YDisplayUpdateCallback callback)
pas procedure registerValueCallback( callback: TGenericUpdateCallback)
vb procedure registerValueCallback( ByVal callback As GenericUpdateCallback)
cs void registerValueCallback( UpdateCallback callback)
java void registerValueCallback( UpdateCallback callback)
py def registerValueCallback( callback)
m -(void) registerValueCallback : (YFunctionUpdateCallback) callback
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**vsourceset\_logicalName()**  
**vsourcesetLogicalName()**  
**vsourceset\_logicalName()**

**YVSource**

Modifie le nom logique de la source de tension.

```
function set_logicalName( ByVal newval As String) As Integer
```

**vsourceset\_logicalName()**  
**vsourcesetLogicalName()**  
**vsourceset\_logicalName()**

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.

**vsouce→set(userData)****YVSource****vsouce→setUserData()|vsouce.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).

```
procedure set(userData( ByVal data As Object)
```

**vsouce→set(userData)****vsouce→setUserData()|vsouce.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

**vsouce→set\_voltage()****YVSource****vsouce→setVoltage()vsouce.set\_voltage()**

Règle la tension de sortie du module (en millivolts).

```
function set_voltage( ByVal newval As Integer) As Integer
```

**vsouce→set\_voltage()****vsouce→setVoltage()vsouce.set\_voltage()**

Règle la tension de sortie du module (en millivolts).

```
js function set_voltage( newval)
```

```
php function set_voltage( $newval)
```

```
cpp int set_voltage( int newval)
```

```
m -(int) setVoltage : (int) newval
```

```
pas function set_voltage( newval: LongInt): integer
```

```
vb function set_voltage( ByVal newval As Integer) As Integer
```

```
cs int set_voltage( int newval)
```

```
java int set_voltage( int newval)
```

```
py def set_voltage( newval)
```

```
cmd YVSource target set_voltage newval
```

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**vsOURCE→voltageMove()vsOURCE.voltageMove()****YVSource**

Déclenche une variation constante de la sortie vers une valeur donnée.

function **voltageMove( ByVal target As Integer,**  
**ByVal ms\_duration As Integer) As Integer**

**vsOURCE→voltageMove()vsOURCE.voltageMove()**

Déclenche une variation constante de la sortie vers une valeur donnée.

**js** function **voltageMove( target, ms\_duration)**  
**php** function **voltageMove( \$target, \$ms\_duration)**  
**cpp** int **voltageMove( int target, int ms\_duration)**  
**m** -(int) **voltageMove : (int) target : (int) ms\_duration**  
**pas** function **voltageMove( target: integer, ms\_duration: integer): integer**  
**vb** function **voltageMove( ByVal target As Integer,**  
                  **ByVal ms\_duration As Integer) As Integer**  
**cs** int **voltageMove( int target, int ms\_duration)**  
**java** int **voltageMove( int target, int ms\_duration)**  
**py** def **voltageMove( target, ms\_duration)**  
**cmd** YVSource target **voltageMove target ms\_duration**

**Paramètres :**

**target** nouvelle valeur de sortie à la fin de la transition, en millivolts.

**ms\_duration** durée de la transition, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.47. Interface de la fonction WakeUpMonitor

La fonction WakeUpMonitor prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_wakeupmonitor.js'></script>
nodejs var yoctolib = require('yoctolib');
var YWakeUpMonitor = yoctolib.YWakeUpMonitor;
php require_once('yocto_wakeupmonitor.php');
cpp #include "yocto_wakeupmonitor.h"
m #import "yocto_wakeupmonitor.h"
pas uses yocto_wakeupmonitor;
vb yocto_wakeupmonitor.vb
cs yocto_wakeupmonitor.cs
java import com.yoctopuce.YoctoAPI.YWakeUpMonitor;
py from yocto_wakeupmonitor import *

```

### Fonction globales

#### yFindWakeUpMonitor(func)

Permet de retrouver un moniteur d'après un identifiant donné.

#### yFirstWakeUpMonitor()

Commence l'énumération des Moniteurs accessibles par la librairie.

### Méthodes des objets YWakeUpMonitor

#### wakeupmonitor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### wakeupmonitor→get\_advertisedValue()

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

#### wakeupmonitor→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

#### wakeupmonitor→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

#### wakeupmonitor→get\_friendlyName()

Retourne un identifiant global du moniteur au format NOM\_MODULE . NOM\_FONCTION.

#### wakeupmonitor→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### wakeupmonitor→get\_functionId()

Retourne l'identifiant matériel du moniteur, sans référence au module.

#### wakeupmonitor→get\_hardwareId()

Retourne l'identifiant matériel unique du moniteur au format SERIAL . FUNCTIONID.

#### wakeupmonitor→get\_logicalName()

Retourne le nom logique du moniteur.

#### wakeupmonitor→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupmonitor→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

### 3. Reference

<b>wakeupmonitor→get_nextWakeUp()</b>
Retourne la prochaine date/heure de réveil agendée (format UNIX)
<b>wakeupmonitor→get_powerDuration()</b>
Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.
<b>wakeupmonitor→get_sleepCountdown()</b>
Retourne le temps avant le prochain sommeil.
<b>wakeupmonitor→get_userData()</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>wakeupmonitor→get_wakeUpReason()</b>
Renvoie la raison du dernier réveil.
<b>wakeupmonitor→get_wakeUpState()</b>
Revoie l'état actuel du moniteur
<b>wakeupmonitor→isOnline()</b>
Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.
<b>wakeupmonitor→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.
<b>wakeupmonitor→load(msValidity)</b>
Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.
<b>wakeupmonitor→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.
<b>wakeupmonitor→nextWakeUpMonitor()</b>
Continue l'énumération des Moniteurs commencée à l'aide de yFirstWakeUpMonitor( ).
<b>wakeupmonitor→registerValueCallback(callback)</b>
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<b>wakeupmonitor→resetSleepCountDown()</b>
Réinitialise le compteur de mise en sommeil.
<b>wakeupmonitor→set_logicalName(newval)</b>
Modifie le nom logique du moniteur.
<b>wakeupmonitor→set_nextWakeUp(newval)</b>
Modifie les jours de la semaine où un réveil doit avoir lieu.
<b>wakeupmonitor→set_powerDuration(newval)</b>
Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.
<b>wakeupmonitor→set_sleepCountdown(newval)</b>
Modifie le temps avant le prochain sommeil .
<b>wakeupmonitor→set_userData(data)</b>
Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).
<b>wakeupmonitor→sleep(secBeforeSleep)</b>
Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.
<b>wakeupmonitor→sleepFor(secUntilWakeUp, secBeforeSleep)</b>
Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.
<b>wakeupmonitor→sleepUntil(wakeUpTime, secBeforeSleep)</b>
Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.
<b>wakeupmonitor→wait_async(callback, context)</b>

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**wakeupmonitor→wakeUp()**

Force un réveil.

## YWakeUpMonitor.FindWakeUpMonitor() yFindWakeUpMonitor()yFindWakeUpMonitor()

YWakeUpMonitor

Permet de retrouver un moniteur d'après un identifiant donné.

```
function yFindWakeUpMonitor( ByVal func As String) As YWakeUpMonitor
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.isOnline()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le moniteur sans ambiguïté

### Retourne :

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

## **YWakeUpMonitor.FirstWakeUpMonitor() yFirstWakeUpMonitor()yFirstWakeUpMonitor()**

## **YWakeUpMonitor**

Commence l'énumération des Moniteurs accessibles par la librairie.

```
function yFirstWakeUpMonitor( ) As YWakeUpMonitor
```

Utiliser la fonction `YWakeUpMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

### **Retourne :**

un pointeur sur un objet `YWakeUpMonitor`, correspondant au premier moniteur accessible en ligne, ou `null` si il n'y a pas de Moniteurs disponibles.

**wakeupmonitor→describe()**  
**wakeupmonitor.describe()****YWakeUpMonitor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**function describe( ) As String**

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le moniteur (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**wakeupmonitor→get\_advertisedValue()**  
**wakeupmonitor→advertisedValue()**  
**wakeupmonitor.get\_advertisedValue()**

**YWakeUpMonitor**

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wakeupmonitor→get\_errorMessage()**  
**wakeupmonitor→errorMessage()**  
**wakeupmonitor.get\_errorMessage()**

**YWakeUpMonitor**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

**wakeupmonitor→get\_errorType()**  
**wakeupmonitor→errorType()**  
**wakeupmonitor.get\_errorType()****YWakeUpMonitor**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

**function get\_errorType( ) As YRETCODE**

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\_functionDescriptor()  
wakeupmonitor→functionDescriptor()  
wakeupmonitor.get\_functionDescriptor()

YWakeUpMonitor

---

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

function **get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**wakeupmonitor→get\_functionId()**  
**wakeupmonitor→functionId()**  
**wakeupmonitor.get\_functionId()****YWakeUpMonitor**

Retourne l'identifiant matériel du moniteur, sans référence au module.

```
function get_functionId( ) As String
```

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.

function **get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moniteur (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le moniteur (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

wakeupmonitor→get\_logicalName()  
wakeupmonitor→logicalName()  
wakeupmonitor.get\_logicalName()

YWakeUpMonitor

Retourne le nom logique du moniteur.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du moniteur.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

wakeupmonitor→get\_module()

YWakeUpMonitor

wakeupmonitor→module()

wakeupmonitor.get\_module()

---

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

**Retourne :**

une instance de YModule

wakeupmonitor→get\_nextWakeUp()

YWakeUpMonitor

wakeupmonitor→nextWakeUp()

wakeupmonitor.get\_nextWakeUp()

---

Retourne la prochaine date/heure de réveil agendée (format UNIX)

```
function get_nextWakeUp( ) As Long
```

**Retourne :**

un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTWAKEUP\_INVALID.

wakeupmonitor→get\_powerDuration()  
wakeupmonitor→powerDuration()  
wakeupmonitor.get\_powerDuration()

YWakeUpMonitor

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
function get_powerDuration( ) As Integer
```

**Retourne :**

un entier représentant le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement

En cas d'erreur, déclenche une exception ou retourne Y\_POWERDURATION\_INVALID.

wakeupmonitor→get\_sleepCountdown()  
wakeupmonitor→sleepCountdown()  
wakeupmonitor.get\_sleepCountdown()

YWakeUpMonitor

Retourne le temps avant le prochain sommeil.

```
function get_sleepCountdown( ) As Integer
```

**Retourne :**

un entier représentant le temps avant le prochain sommeil

En cas d'erreur, déclenche une exception ou retourne Y\_SLEEP\_COUNTDOWN\_INVALID.

**wakeupmonitor→get(userData)**  
**wakeupmonitor→userData()**  
**wakeupmonitor.get(userData)**

**YWakeUpMonitor**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**wakeupmonitor→get\_wakeUpReason()**  
**wakeupmonitor→wakeUpReason()**  
**wakeupmonitor.get\_wakeUpReason()**

**YWakeUpMonitor**

Renvoie la raison du dernier réveil.

```
function get_wakeUpReason( ) As Integer
```

**Retourne :**

une valeur parmi Y\_WAKEUPREASON\_USBPOWER, Y\_WAKEUPREASON\_EXTPOWER,  
Y\_WAKEUPREASON\_ENDOFSLEEP, Y\_WAKEUPREASON\_EXTSIG1,  
Y\_WAKEUPREASON\_SCHEDULE1 et Y\_WAKEUPREASON\_SCHEDULE2

En cas d'erreur, déclenche une exception ou retourne Y\_WAKEUPREASON\_INVALID.

wakeupmonitor→get\_wakeUpState()  
wakeupmonitor→wakeUpState()  
wakeupmonitor.get\_wakeUpState()

YWakeUpMonitor

Revoie l'état actuel du moniteur

```
function get_wakeUpState( ) As Integer
```

**Retourne :**

soit Y\_WAKEUPSTATE\_SLEEPING, soit Y\_WAKEUPSTATE\_AWAKE

En cas d'erreur, déclenche une exception ou retourne Y\_WAKEUPSTATE\_INVALID.

**wakeupmonitor→isOnline()wakeupmonitor.isOnline()****YWakeUpMonitor**

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du moniteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le moniteur est joignable, false sinon

**wakeupmonitor→load()wakeupmonitor.load()****YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→nextWakeUpMonitor()**  
**wakeupmonitor.nextWakeUpMonitor()****YWakeUpMonitor**

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

```
function nextWakeUpMonitor( ) As YWakeUpMonitor
```

**Retourne :**

un pointeur sur un objet `YWakeUpMonitor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wakeupmonitor→registerValueCallback()**  
**wakeupmonitor.registerValueCallback()****YWakeUpMonitor**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**wakeupmonitor→resetSleepCountDown()**  
**wakeupmonitor.resetSleepCountDown()****YWakeUpMonitor**

Réinitialise le compteur de mise en sommeil.

```
function resetSleepCountDown( ) As Integer
```

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set\_logicalName()  
wakeupmonitor→setLogicalName()  
wakeupmonitor.set\_logicalName()

YWakeUpMonitor

Modifie le nom logique du moniteur.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

**newval** une chaîne de caractères représentant le nom logique du moniteur.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set\_nextWakeUp()**  
wakeupmonitor→**setNextWakeUp()**  
**wakeupmonitor.set\_nextWakeUp()**

**YWakeUpMonitor**

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
function set_nextWakeUp( ByVal newval As Long) As Integer
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set\_powerDuration()  
wakeupmonitor→setPowerDuration()  
wakeupmonitor.set\_powerDuration()

YWakeUpMonitor

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

function **set\_powerDuration( ByVal newval As Integer) As Integer**

**Paramètres :**

**newval** un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→set\_sleepCountdown()**  
**wakeupmonitor→setSleepCountdown()**  
**wakeupmonitor.set\_sleepCountdown()**

**YWakeUpMonitor**

Modifie le temps avant le prochain sommeil .

```
function set_sleepCountdown( ByVal newval As Integer) As Integer
```

**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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

**wakeupmonitor→sleep()wakeupmonitor.sleep()****YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
function sleep( ) As Integer
```

**Paramètres :**

**secBeforeSleep** nombre de seconde avant la mise en sommeil

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→sleepFor()  
wakeupmonitor.sleepFor()****YWakeUpMonitor**

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

**function sleepFor( ) As Integer**

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

**Paramètres :****secUntilWakeUp** nombre de secondes avant le prochain réveil**secBeforeSleep** nombre de secondes avant la mise en sommeil**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→sleepUntil()**  
**wakeupmonitor.sleepUntil()****YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

function **sleepUntil( ) As Integer**

Le compte à rebours avant la mise en sommeil peut être annulé grâce à `resetSleepCountDown`.

**Paramètres :**

**wakeUpTime** date/heure du réveil (format UNIX)

**secBeforeSleep** nombre de secondes avant la mise en sommeil

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupmonitor→wakeUp()wakeupmonitor.wakeUp()**

**YWakeUpMonitor**

Force un réveil.

```
function wakeUp( ) As Integer
```

## 3.48. Interface de la fonction WakeUpSchedule

La fonction WakeUpSchedule implémente une condition de réveil. Le réveil est spécifiée par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```

js <script type='text/javascript' src='yocto_wakeupschedule.js'></script>
nodejs var yoctolib = require('yoctolib');
var YWakeUpSchedule = yoctolib.YWakeUpSchedule;
php require_once('yocto_wakeupschedule.php');
cpp #include "yocto_wakeupschedule.h"
m #import "yocto_wakeupschedule.h"
pas uses yocto_wakeupschedule;
vb yocto_wakeupschedule.vb
cs yocto_wakeupschedule.cs
java import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py from yocto_wakeupschedule import *

```

### Fonction globales

#### yFindWakeUpSchedule(func)

Permet de retrouver un réveil agendé d'après un identifiant donné.

#### yFirstWakeUpSchedule()

Commence l'énumération des réveils agendés accessibles par la librairie.

### Méthodes des objets YWakeUpSchedule

#### wakeupschedule→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### wakeupschedule→get\_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

#### wakeupschedule→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

#### wakeupschedule→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

#### wakeupschedule→get\_friendlyName()

Retourne un identifiant global du réveil agendé au format NOM\_MODULE . NOM\_FONCTION.

#### wakeupschedule→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### wakeupschedule→get\_functionId()

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

#### wakeupschedule→get\_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format SERIAL . FUNCTIONID.

#### wakeupschedule→get\_hours()

Retourne les heures où le réveil est actif..

#### wakeupschedule→get\_logicalName()

Retourne le nom logique du réveil agendé.

#### wakeupschedule→get\_minutes()

Retourne toutes les minutes de chaque heure où le réveil est actif.

#### wakeupschedule→get\_minutesA()

### 3. Reference

Retourne les minutes de l'intervalle 00-29 de chaque heure où le réveil est actif.

#### wakeupschedule→get\_minutesB()

Retourne les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif.

#### wakeupschedule→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupschedule→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wakeupschedule→get\_monthDays()

Retourne les jours du mois où le réveil est actif..

#### wakeupschedule→get\_months()

Retourne les mois où le réveil est actif..

#### wakeupschedule→get\_nextOccurrence()

Retourne la date/heure de la prochaine occurrence de réveil

#### wakeupschedule→get\_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### wakeupschedule→get\_weekDays()

Retourne les jours de la semaine où le réveil est actif..

#### wakeupschedule→isOnline()

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

#### wakeupschedule→isOnline\_async(callback, context)

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

#### wakeupschedule→load(msValidity)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

#### wakeupschedule→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

#### wakeupschedule→nextWakeUpSchedule()

Continue l'énumération des réveils agendés commencée à l'aide de yFirstWakeUpSchedule().

#### wakeupschedule→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### wakeupschedule→set\_hours(newval, newval)

Modifie les heures où un réveil doit avoir lieu

#### wakeupschedule→set\_logicalName(newval)

Modifie le nom logique du réveil agendé.

#### wakeupschedule→set\_minutes(bitmap)

Modifie toutes les minutes où un réveil doit avoir lieu

#### wakeupschedule→set\_minutesA(newval, newval)

Modifie les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

#### wakeupschedule→set\_minutesB(newval)

Modifie les minutes de l'intervalle 30-59 où un réveil doit avoir lieu.

#### wakeupschedule→set\_monthDays(newval, newval)

Modifie les jours du mois où un réveil doit avoir lieu

#### wakeupschedule→set\_months(newval, newval)

Modifie les mois où un réveil doit avoir lieu

#### wakeupschedule→set\_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**wakeupschedule→set\_weekDays(newval, newval)**

Modifie les jours de la semaine où un réveil doit avoir lieu

**wakeupschedule→wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YWakeUpSchedule.FindWakeUpSchedule() yFindWakeUpSchedule()yFindWakeUpSchedule()

### YWakeUpSchedule

Permet de retrouver un réveil agendé d'après un identifiant donné.

```
function yFindWakeUpSchedule( ByVal func As String) As YWakeUpSchedule
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le réveil agendé soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpSchedule.isOnLine()` pour tester si le réveil agendé est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

#### Paramètres :

`func` une chaîne de caractères qui référence le réveil agendé sans ambiguïté

#### Retourne :

un objet de classe `YWakeUpSchedule` qui permet ensuite de contrôler le réveil agendé.

**YWakeUpSchedule.FirstWakeUpSchedule()****yFirstWakeUpSchedule()yFirstWakeUpSchedule()****YWakeUpSchedule**

Commence l'énumération des réveils agendés accessibles par la librairie.

```
function yFirstWakeUpSchedule( ) As YWakeUpSchedule
```

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule()` pour itérer sur les autres réveils agendés.

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule`, correspondant au premier réveil agendé accessible en ligne, ou `null` si il n'y a pas de réveils agendés disponibles.

**wakeupschedule→describe()**  
**wakeupschedule.describe()****YWakeUpSchedule**

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format TYPE ( NAME )=SERIAL . FUNCTIONID.

**function describe( ) As String**

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 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).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante du réveil agendé (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wakeupschedule→get\_errorMessage()**  
**wakeupschedule→errorMessage()**  
**wakeupschedule.get\_errorMessage()**

**YWakeUpSchedule**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

**wakeupschedule→get\_errorType()**  
**wakeupschedule→errorType()**  
**wakeupschedule.get\_errorType()**

**YWakeUpSchedule**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()  
wakeupschedule→functionDescriptor()  
wakeupschedule.get\_functionDescriptor()

YWakeUpSchedule

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

function **get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**wakeupschedule→get\_functionId()**  
**wakeupschedule→functionId()**  
**wakeupschedule.get\_functionId()**

**YWakeUpSchedule**

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

```
function get_functionId( ) As String
```

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le réveil agendé (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**wakeupschedule→get\_hardwareId()**  
**wakeupschedule→hardwareId()**  
**wakeupschedule.get\_hardwareId()**

**YWakeUpSchedule**

Retourne l'identifiant matériel unique du réveil agendé au format SERIAL.FUNCTIONID.

**function get\_hardwareId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du réveil agendé (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le réveil agendé (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

wakeupschedule→get\_hours()  
wakeupschedule→hours()  
wakeupschedule.get\_hours()

YWakeUpSchedule

Retourne les heures où le réveil est actif..

```
function get_hours( ) As Integer
```

**Retourne :**

un entier représentant les heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_HOURS\_INVALID.

wakeupschedule→get\_logicalName()  
wakeupschedule→logicalName()  
wakeupschedule.get\_logicalName()

---

YWakeUpSchedule

Retourne le nom logique du réveil agendé.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du réveil agendé.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

wakeupschedule→get\_minutes()  
wakeupschedule→minutes()  
wakeupschedule.get\_minutes()

YWakeUpSchedule

Retourne toutes les minutes de chaque heure où le réveil est actif.

```
function get_minutes( ) As Long
```

**wakeupschedule→get\_minutesA()**  
**wakeupschedule→minutesA()**  
**wakeupschedule.get\_minutesA()**

**YWakeUpSchedule**

---

Retourne les minutes de l'intervalle 00-29 de chaque heures où le réveil est actif.

```
function get_minutesA( ) As Integer
```

**Retourne :**

un entier représentant les minutes de l'intervalle 00-29 de chaque heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MINUTESA\_INVALID.

wakeupschedule→get\_minutesB()  
wakeupschedule→minutesB()  
wakeupschedule.get\_minutesB()

YWakeUpSchedule

Retourne les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif.

function **get\_minutesB( )** As Integer

**Retourne :**

un entier représentant les minutes de l'intervalle 30-59 de chaque heure où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MINUTESB\_INVALID.

**wakeupschedule→get\_module()**  
**wakeupschedule→module()**  
**wakeupschedule.get\_module()**

---

**YWakeUpSchedule**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

**function get\_module( ) As YModule**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

wakeupschedule→get\_monthDays()

YWakeUpSchedule

wakeupschedule→monthDays()

wakeupschedule.get\_monthDays()

---

Retourne les jours du mois où le réveil est actif..

```
function get_monthDays( ) As Integer
```

**Retourne :**

un entier représentant les jours du mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MONTHDAYS\_INVALID.

wakeupschedule→get\_months()

YWakeUpSchedule

wakeupschedule→months()

wakeupschedule.get\_months()

---

Retourne les mois où le réveil est actif..

```
function get_months( ) As Integer
```

**Retourne :**

un entier représentant les mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_MONTHS\_INVALID.

wakeupschedule→get\_nextOccurence()  
wakeupschedule→nextOccurence()  
wakeupschedule.get\_nextOccurence()

YWakeUpSchedule

Retourne la date/heure de la prochaine occurence de réveil

```
function get_nextOccurence( ) As Long
```

**Retourne :**

un entier représentant la date/heure de la prochaine occurence de réveil

En cas d'erreur, déclenche une exception ou retourne Y\_NEXTOCCURENCE\_INVALID.

wakeupschedule→get(userData)

YWakeUpSchedule

wakeupschedule→userData()

wakeupschedule.get(userData)

---

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

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..

```
function get_weekDays( ) As Integer
```

**Retourne :**

un entier représentant les jours de la semaine où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y\_WEEKDAYS\_INVALID.

**wakeupschedule→isOnline()**  
**wakeupschedule.isOnline()**

---

**YWakeUpSchedule**

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache du réveil agendé sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le réveil agendé est joignable, false sinon

**wakeupschedule→load()wakeupschedule.load()****YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→nextWakeUpSchedule()**  
**wakeupschedule.nextWakeUpSchedule()**

**YWakeUpSchedule**

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

```
function nextWakeUpSchedule( ) As YWakeUpSchedule
```

**Retourne :**

un pointeur sur un objet `YWakeUpSchedule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wakeupschedule→registerValueCallback()  
wakeupschedule.registerValueCallback()****YWakeUpSchedule**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wakeupschedule→set\_hours()  
wakeupschedule→setHours()  
wakeupschedule.set\_hours()

YWakeUpSchedule

Modifie les heures où un réveil doit avoir lieu

function **set\_hours**( ByVal **newval** As Integer) As Integer

**Paramètres :**

**newval** un entier représentant les heures où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_logicalName()**  
**wakeupschedule→setLogicalName()**  
**wakeupschedule.set\_logicalName()**

**YWakeUpSchedule**

Modifie le nom logique du réveil agendé.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du réveil agendé.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_minutes()**  
**wakeupschedule→setMinutes()**  
**wakeupschedule.set\_minutes()**

**YWakeUpSchedule**

Modifie toutes les minutes où un réveil doit avoir lieu

function **set\_minutes( ) As Integer**

**Paramètres :**

**bitmap** Minutes 00-59 de chaque heure où le réveil est actif.

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_minutesA()  
wakeupschedule→setMinutesA()  
wakeupschedule.set\_minutesA()

YWakeUpSchedule

Modifie les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

```
function set_minutesA( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant les minutes de l'intervalle 00-29 où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wakeupschedule→set\_minutesB()**  
**wakeupschedule→setMinutesB()**  
**wakeupschedule.set\_minutesB()**

**YWakeUpSchedule**

Modifie les minutes de l'intervalle 30-59 où un réveil doit avoir lieu.

```
function set_minutesB( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant les minutes de l'intervalle 30-59 où un réveil doit avoir lieu

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_monthDays()  
wakeupschedule→setMonthDays()  
wakeupschedule.set\_monthDays()

YWakeUpSchedule

Modifie les jours du mois où un réveil doit avoir lieu

```
function set_monthDays( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant les jours du mois où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set\_months()  
wakeupschedule→setMonths()  
wakeupschedule.set\_months()

YWakeUpSchedule

Modifie les mois où un réveil doit avoir lieu

```
function set_months( ByVal newval As Integer) As Integer
```

**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).

procedure **set(userData)** ByVal **data** As Object

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

wakeupschedule→set\_weekDays()  
wakeupschedule→setWeekDays()  
wakeupschedule.set\_weekDays()

YWakeUpSchedule

---

Modifie les jours de la semaine où un réveil doit avoir lieu

```
function set_weekDays( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** un entier représentant les jours de la semaine où un réveil doit avoir lieu

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## 3.49. Interface de la fonction Watchdog

La fonction WatchDog est gérée comme un relais qui couperait brièvement l'alimentation d'un appareil après un d'attente temps donné afin de provoquer une réinitialisation complète de cet appareil. Il suffit d'appeler le watchdog à intervalle régulier pour l'empêcher de provoquer la réinitialisation. Le watchdog peut aussi être piloté directement à l'aide des méthodes *pulse* et *delayedpulse* pour éteindre un appareil pendant un temps donné.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_watchdog.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YWatchdog = yoctolib.YWatchdog;
php	require_once('yocto_watchdog.php');
cpp	#include "yocto_watchdog.h"
m	#import "yocto_watchdog.h"
pas	uses yocto_watchdog;
vb	yocto_watchdog.vb
cs	yocto_watchdog.cs
java	import com.yoctopuce.YoctoAPI.YWatchdog;
py	from yocto_watchdog import *

### Fonction globales

#### yFindWatchdog(func)

Permet de retrouver un watchdog d'après un identifiant donné.

#### yFirstWatchdog()

Commence l'énumération des watchdog accessibles par la librairie.

### Méthodes des objets YWatchdog

#### watchdog->delayedPulse(ms\_delay, ms\_duration)

Pré-programme une impulsion

#### watchdog->describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format TYPE ( NAME ) = SERIAL . FUNCTIONID.

#### watchdog->get\_advertisedValue()

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

#### watchdog->get\_autoStart()

Retourne l'état du watchdog à la mise sous tension du module.

#### watchdog->get\_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

#### watchdog->get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

#### watchdog->get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

#### watchdog->get\_friendlyName()

Retourne un identifiant global du watchdog au format NOM\_MODULE . NOM\_FONCTION.

#### watchdog->get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### watchdog->get\_functionId()

Retourne l'identifiant matériel du watchdog, sans référence au module.
<b>watchdog→get_hardwareId()</b>
Retourne l'identifiant matériel unique du watchdog au format SERIAL . FUNCTIONID.
<b>watchdog→get_logicalName()</b>
Retourne le nom logique du watchdog.
<b>watchdog→get_maxTimeOnStateA()</b>
Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.
<b>watchdog→get_maxTimeOnStateB()</b>
Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.
<b>watchdog→get_module()</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>watchdog→get_module_async(callback, context)</b>
Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
<b>watchdog→get_output()</b>
Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.
<b>watchdog→get_pulseTimer()</b>
Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.
<b>watchdog→get_running()</b>
Retourne l'état du watchdog.
<b>watchdog→get_state()</b>
Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).
<b>watchdog→get_stateAtPowerOn()</b>
Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).
<b>watchdog→get_triggerDelay()</b>
Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.
<b>watchdog→get_triggerDuration()</b>
Retourne la durée d'un reset généré par le watchdog, en millisecondes.
<b>watchdog→get(userData)</b>
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).
<b>watchdog→isOnline()</b>
Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.
<b>watchdog→isOnline_async(callback, context)</b>
Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.
<b>watchdog→load(msValidity)</b>
Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.
<b>watchdog→load_async(msValidity, callback, context)</b>
Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.
<b>watchdog→nextWatchdog()</b>
Continue l'énumération des watchdog commencée à l'aide de yFirstWatchdog( ).
<b>watchdog→pulse(ms_duration)</b>
Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

**watchdog->registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

**watchdog->resetWatchdog()**

Réinitialise le WatchDog.

**watchdog->set\_autoStart(newval)**

Modifie l'état du watching au démarrage du module.

**watchdog->set\_logicalName(newval)**

Modifie le nom logique du watchdog.

**watchdog->set\_maxTimeOnStateA(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

**watchdog->set\_maxTimeOnStateB(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

**watchdog->set\_output(newval)**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

**watchdog->set\_running(newval)**

Modifie manuellement l'état de fonctionnement du watchdog.

**watchdog->set\_state(newval)**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

**watchdog->set\_stateAtPowerOn(newval)**

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

**watchdog->set\_triggerDelay(newval)**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

**watchdog->set\_triggerDuration(newval)**

Modifie la durée des resets générés par le watchdog, en millisecondes.

**watchdog->set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get(userData).

**watchdog->wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

## YWatchdog.FindWatchdog() yFindWatchdog()yFindWatchdog()

**YWatchdog**

Permet de retrouver un watchdog d'après un identifiant donné.

```
function yFindWatchdog( ByVal func As String) As YWatchdog
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le watchdog soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWatchdog.isOnLine()` pour tester si le watchdog est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

### Paramètres :

`func` une chaîne de caractères qui référence le watchdog sans ambiguïté

### Retourne :

un objet de classe `YWatchdog` qui permet ensuite de contrôler le watchdog.

## YWatchdog.FirstWatchdog() yFirstWatchdog()yFirstWatchdog()

**YWatchdog**

Commence l'énumération des watchdog accessibles par la librairie.

```
function yFirstWatchdog( ) As YWatchdog
```

Utiliser la fonction `YWatchdog.nextWatchdog( )` pour itérer sur les autres watchdog.

**Retourne :**

un pointeur sur un objet `YWatchdog`, correspondant au premier watchdog accessible en ligne, ou `null` si il n'y a pas de watchdog disponibles.

## watchdog→delayedPulse() watchdog.delayedPulse()

YWatchdog

Pré-programme une impulsion

```
function delayedPulse( ByVal ms_delay As Integer,  
                      ByVal ms_duration As Integer) As Integer
```

### Paramètres :

**ms\_delay** délai d'attente avant l'impulsion, en millisecondes

**ms\_duration** durée de l'impulsion, en millisecondes

### Retourne :

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→describe()watchdog.describe()****YWatchdog**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format TYPE ( NAME )=SERIAL.FUNCTIONID.

```
function describe( ) As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant le watchdog (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**watchdog→get\_advertisedValue()**  
**watchdog→advertisedValue()**  
**watchdog.get\_advertisedValue()**

**YWatchdog**

---

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**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\_ADVISEDVALUE\_INVALID.

**watchdog→get\_autoStart()****YWatchdog****watchdog→autoStart()watchdog.get\_autoStart()**

Retourne l'état du watchdog à la mise sous tension du module.

```
function get_autoStart( ) As Integer
```

**Retourne :**

soit Y\_AUTOSTART\_OFF, soit Y\_AUTOSTART\_ON, selon l'état du watchdog à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y\_AUTOSTART\_INVALID.

**watchdog→get\_countdown()**

**YWatchdog**

**watchdog→countdown()watchdog.get\_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

```
function get_countdown( ) As Long
```

Si aucune impulsion n'est programmée, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne Y\_COUNTDOWN\_INVALID.

**watchdog→get\_errorMessage()**  
**watchdog→errorMessage()**  
**watchdog.get\_errorMessage()****YWatchdog**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
function get_errorMessage( ) As String
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

**watchdog→get\_errorType()**

**YWatchdog**

**watchdog→errorType()watchdog.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**watchdog→functionDescriptor()**  
**watchdog.get\_functionDescriptor()**

**YWatchdog**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

```
function get_functionDescriptor( ) As YFUN_DESCR
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera  
Y\_FUNCTIONDESCRIPTOR\_INVALID

**watchdog→get\_functionId()**

**YWatchdog**

**watchdog→functionId()watchdog.get\_functionId()**

---

Retourne l'identifiant matériel du watchdog, sans référence au module.

**function get\_functionId( ) As String**

Par exemple `relay1`.

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

**watchdog→get\_hardwareId()****YWatchdog****watchdog→hardwareId()watchdog.get\_hardwareId()**

Retourne l'identifiant matériel unique du watchdog au format SERIAL.FUNCTIONID.

```
function get_hardwareId( ) As String
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du watchdog (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant le watchdog (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**watchdog→get\_logicalName()**  
**watchdog→logicalName()**  
**watchdog.get\_logicalName()**

---

**YWatchdog**

Retourne le nom logique du watchdog.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique du watchdog.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**watchdog→get\_maxTimeOnStateA()****YWatchdog****watchdog→maxTimeOnStateA()****watchdog.get\_maxTimeOnStateA()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
function get_maxTimeOnStateA( ) As Long
```

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.

**function get\_maxTimeOnStateB( ) As Long**

Zéro signifie qu'il n'y a pas de limitation

**Retourne :**

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne Y\_MAXTIMEONSTATEB\_INVALID.

**watchdog→get\_module()****YWatchdog****watchdog→module()watchdog.get\_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module() As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**watchdog→get\_output()**

**YWatchdog**

**watchdog→output()watchdog.get\_output()**

---

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
function get_output( ) As Integer
```

**Retourne :**

soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y\_OUTPUT\_INVALID.

**watchdog→get\_pulseTimer()****YWatchdog****watchdog→pulseTimer()watchdog.get\_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
function get_pulseTimer( ) As Long
```

Si aucune impulsion n'est en cours, retourne zéro.

**Retourne :**

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne Y\_PULSE\_TIMER\_INVALID.

**watchdog→get\_running()**

**YWatchdog**

**watchdog→running()watchdog.get\_running()**

---

Retourne l'état du watchdog.

```
function get_running( ) As Integer
```

**Retourne :**

soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon l'état du watchdog

En cas d'erreur, déclenche une exception ou retourne Y\_RUNNING\_INVALID.

**watchdog→get\_state()****YWatchdog****watchdog→state()watchdog.get\_state()**

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
function get_state( ) As Integer
```

**Retourne :**

soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y\_STATE\_INVALID.

**watchdog→get\_stateAtPowerOn()**  
**watchdog→stateAtPowerOn()**  
**watchdog.get\_stateAtPowerOn()**

**YWatchdog**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

function **get\_stateAtPowerOn( ) As Integer**

**Retourne :**

une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B représentant l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y\_STATEATPOWERON\_INVALID.

**watchdog→get\_triggerDelay()**  
**watchdog→triggerDelay()**  
**watchdog.get\_triggerDelay()****YWatchdog**

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

```
function get_triggerDelay( ) As Long
```

**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.

<b>watchdog→get_triggerDuration()</b>	<b>YWatchdog</b>
<b>watchdog→triggerDuration()</b>	
<b>watchdog.get_triggerDuration()</b>	

---

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

```
function get_triggerDuration( ) As Long
```

**Retourne :**

un entier représentant la durée d'un reset généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y\_TRIGGER\_DURATION\_INVALID.

**watchdog→get(userData)****YWatchdog****watchdog→userData()watchdog.get(userData)**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**watchdog→isOnline()watchdog.isOnline()****YWatchdog**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
function isOnline( ) As Boolean
```

Si les valeurs des attributs en cache du watchdog sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si le watchdog est joignable, false sinon

**watchdog→load()watchdog.load()****YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→nextWatchdog()**  
**watchdog.nextWatchdog()**

---

**YWatchdog**

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

function **nextWatchdog( ) As YWatchdog**

**Retourne :**

un pointeur sur un objet `YWatchdog` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**watchdog→pulse()watchdog.pulse()****YWatchdog**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
function pulse( ByVal ms_duration As Integer) As Integer
```

**Paramètres :**

**ms\_duration** durée de l'impulsion, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→registerValueCallback()  
watchdog.registerValueCallback()****YWatchdog**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**watchdog→resetWatchdog()**  
**watchdog.resetWatchdog()****YWatchdog**

Réinitialise le WatchDog.

```
function resetWatchdog( ) As Integer
```

Quand le watchdog est en fonctionnement cette fonction doit être appelée à interval régulier, pour empêcher que le watdog ne se déclenche

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## watchdog→set\_autoStart()

YWatchdog

## watchdog→setAutoStart()watchdog.set\_autoStart()

Modifie l'état du watching au démarrage du module.

```
function set_autoStart( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

### Paramètres :

**newval** soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watching au démarrage du module

### Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_logicalName()**  
**watchdog→setLogicalName()**  
**watchdog.set\_logicalName()**

**YWatchdog**

Modifie le nom logique du watchdog.

```
function set_logicalName( ByVal newval As String) As Integer
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

**Paramètres :**

`newval` une chaîne de caractères représentant le nom logique du watchdog.

**Retourne :**

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_maxTimeOnStateA()**  
**watchdog→setMaxTimeOnStateA()**  
**watchdog.set\_maxTimeOnStateA()**

**YWatchdog**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

function **set\_maxTimeOnStateA( ByVal newval As Long) As Integer**

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_maxTimeOnStateB()**  
**watchdog→setMaxTimeOnStateB()**  
**watchdog.set\_maxTimeOnStateB()**

**YWatchdog**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

function **set\_maxTimeOnStateB( ByVal newval As Long) As Integer**

Zéro signifie qu'il n'y a pas de limitation

**Paramètres :**

**newval** un entier

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **watchdog→set\_output()**

**YWatchdog**

## **watchdog→setOutput()watchdog.set\_output()**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
function set_output( ByVal newval As Integer) As Integer
```

### **Paramètres :**

**newval** soit Y\_OUTPUT\_OFF, soit Y\_OUTPUT\_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_running()****YWatchdog****watchdog→setRunning()watchdog.set\_running()**

Modifie manuellement l'état de fonctionnement du watchdog.

```
function set_running( ByVal newval As Integer) As Integer
```

**Paramètres :**

**newval** soit Y\_RUNNING\_OFF, soit Y\_RUNNING\_ON, selon manuellement l'état de fonctionnement du watchdog

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

## **watchdog→set\_state()**

**YWatchdog**

## **watchdog→setState()watchdog.set\_state()**

---

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
function set_state( ByVal newval As Integer) As Integer
```

### **Paramètres :**

**newval** soit Y\_STATE\_A, soit Y\_STATE\_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

### **Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_stateAtPowerOn()**  
**watchdog→setStateAtPowerOn()**  
**watchdog.set\_stateAtPowerOn()**

**YWatchdog**

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function set_stateAtPowerOn( ByVal newval As Integer) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

**Paramètres :**

**newval** une valeur parmi Y\_STATEATPOWERON\_UNCHANGED, Y\_STATEATPOWERON\_A et Y\_STATEATPOWERON\_B

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDelay()**  
**watchdog→setTriggerDelay()**  
**watchdog.set\_triggerDelay()**

**YWatchdog**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

```
function set_triggerDelay( ByVal newval As Long) As Integer
```

**Paramètres :**

**newval** un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**watchdog→set\_triggerDuration()**  
**watchdog→setTriggerDuration()**  
**watchdog.set\_triggerDuration()**

**YWatchdog**

Modifie la durée des resets générés par le watchdog, en millisecondes.

```
function set_triggerDuration( ByVal newval As Long) As Integer
```

**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).

```
procedure set(userData( ByVal data As Object)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Paramètres :**

**data** objet quelconque à mémoriser

## 3.50. Interface de la fonction Wireless

La fonction YWireless permet de configurer et de contrôler la configuration du réseau sans fil sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wireless.js'></script>
node.js	var yoctolib = require('yoctolib');
	var YWireless = yoctolib.YWireless;
php	require_once('yocto_wireless.php');
cpp	#include "yocto_wireless.h"
m	#import "yocto_wireless.h"
pas	uses yocto_wireless;
vb	yocto_wireless.vb
cs	yocto_wireless.cs
java	import com.yoctopuce.YoctoAPI.YWireless;
py	from yocto_wireless import *

### Fonction globales

#### yFindWireless(func)

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

#### yFirstWireless()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

### Méthodes des objets YWireless

#### wireless→adhocNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

#### wireless→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE ( NAME )=SERIAL . FUNCTIONID.

#### wireless→get\_advertisedValue()

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

#### wireless→get\_channel()

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

#### wireless→get\_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

#### wireless→get\_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

#### wireless→get\_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

#### wireless→get\_friendlyName()

Retourne un identifiant global de l'interface réseau sans fil au format NOM\_MODULE . NOM\_FONCTION.

#### wireless→get\_functionDescriptor()

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

#### wireless→get\_functionId()

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

#### wireless→get\_hardwareId()

### 3. Reference

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

#### wireless→get\_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

#### wireless→get\_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

#### wireless→get\_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

#### wireless→get\_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wireless→get\_module\_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### wireless→get\_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

#### wireless→get\_ssid()

Retourne le nom (SSID) du réseau sans-fil sélectionné.

#### wireless→get\_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

#### wireless→isOnline()

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

#### wireless→isOnline\_async(callback, context)

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

#### wireless→joinNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

#### wireless→load(msValidity)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

#### wireless→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

#### wireless→nextWireless()

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de yFirstWireless( ).

#### wireless→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

#### wireless→set\_logicalName(newval)

Modifie le nom logique de l'interface réseau sans fil.

#### wireless→set\_userData(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→softAPNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

#### wireless→wait\_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YWireless.FindWireless()****YWireless****yFindWireless()yFindWireless()**

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

```
function yFindWireless( ByVal func As String) As YWireless
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau sans fil soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWireless.isOnLine()` pour tester si l'interface réseau sans fil est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

**Paramètres :**

**func** une chaîne de caractères qui référence l'interface réseau sans fil sans ambiguïté

**Retourne :**

un objet de classe `YWireless` qui permet ensuite de contrôler l'interface réseau sans fil.

## **YWireless.FirstWireless() yFirstWireless()yFirstWireless()**

---

**YWireless**

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

```
function yFirstWireless( ) As YWireless
```

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

**Retourne :**

un pointeur sur un objet `YWireless`, correspondant à la première interface réseau sans fil accessible en ligne, ou `null` si il n'y a pas de interfaces réseau sans fil disponibles.

**wireless→adhocNetwork()wireless.adhocNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

function **adhocNetwork( ) As Integer**

Sur le YoctoHub-Wireless-g, il est recommandé d'utiliser de préférence la fonction softAPNetwork() qui crée un pseudo point d'accès, plus efficace et mieux supporté qu'un réseau ad-hoc.

Si une clef d'accès est configurée pour un réseau ad-hoc, le réseau sera protégé par une sécurité WEP40 (5 caractères ou 10 chiffres hexadécimaux) ou WEP128 (13 caractères ou 26 chiffres hexadécimaux). Pour réduire les risques d'intrusion, il est recommandé d'utiliser une clé WEP128 basée sur 26 chiffres hexadécimaux provenant d'une bonne source aléatoire.

N'oubliez pas d'appeler la méthode saveToFlash( ) et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à créer

**securityKey** clé d'accès de réseau, sous forme de chaîne de caractères

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→describe(wireless.describe())****YWireless**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE ( NAME )=SERIAL . FUNCTIONID.

```
function describe() As String
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisée dans un debuggeur.

**Retourne :**

une chaîne de caractères décrivant l'interface réseau sans fil (ex:  
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

**wireless→get\_advertisedValue()**  
**wireless→advertisedValue()**  
**wireless.get\_advertisedValue()**

**YWireless**

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

```
function get_advertisedValue( ) As String
```

**Retourne :**

une chaîne de caractères représentant la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y\_ADVERTISEDVALUE\_INVALID.

**wireless→get\_channel()**

**YWireless**

**wireless→channel()wireless.get\_channel()**

---

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

**function get\_channel( ) As Integer**

**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.

**function get\_detectedWlans( ) As List**

La liste n'est pas mise à jour quand le module est déjà connecté à un accès sans fil (mode "infrastructure"). Pour forcer la détection des réseaux sans fil, il faut appeler addhocNetwork( ) pour se déconnecter du réseau actuel. L'appelant est responsable de la désallocation de la liste retournée.

**Retourne :**

une liste d'objets YWlanRecord, contenant le SSID, le canal, la qualité du signal, et l'algorithme de sécurité utilisé par le réseau sans-fil

En cas d'erreur, déclenche une exception ou retourne une liste vide.

**wireless→get\_errorMessage()**  
**wireless→errorMessage()**  
**wireless.get\_errorMessage()**

**YWireless**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

**function get\_errorMessage( ) As String**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

**Retourne :**

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

**wireless→get\_errorType()****YWireless****wireless→errorType()wireless.get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

```
function get_errorType( ) As YRETCODE
```

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\_functionDescriptor()**  
**wireless→functionDescriptor()**  
**wireless.get\_functionDescriptor()**

---

**YWireless**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

**function get\_functionDescriptor( ) As YFUN\_DESCR**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

**Retourne :**

un identifiant de type YFUN\_DESCR.

Si la fonction n'a jamais été contactée, la valeur renournée sera Y\_FUNCTIONDESCRIPTOR\_INVALID

**wireless→get\_functionId()****YWireless****wireless→functionId()wireless.get\_functionId()**

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

```
function get_functionId( ) As String
```

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\_hwId()****YWireless****wireless→hardwareId()wireless.get\_hwId()**

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

**function get\_hwId( ) As String**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau sans fil (par exemple RELAYL01-123456.relay1).

**Retourne :**

une chaîne de caractères identifiant l'interface réseau sans fil (ex: RELAYL01-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y\_HARDWAREID\_INVALID.

**wireless→get\_linkQuality()****YWireless****wireless→linkQuality()wireless.get\_linkQuality()**

Retourne la qualité de la connection, exprimée en pourcents.

```
function get_linkQuality( ) As Integer
```

**Retourne :**

un entier représentant la qualité de la connection, exprimée en pourcents

En cas d'erreur, déclenche une exception ou retourne Y\_LINKQUALITY\_INVALID.

**wireless→get\_logicalName()**  
**wireless→logicalName()wireless.get\_logicalName()**

---

**YWireless**

Retourne le nom logique de l'interface réseau sans fil.

```
function get_logicalName( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

En cas d'erreur, déclenche une exception ou retourne Y\_LOGICALNAME\_INVALID.

**wireless→get\_message()****YWireless****wireless→message()wireless.get\_message()**

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

```
function get_message( ) As String
```

**Retourne :**

une chaîne de caractères représentant le dernier message de diagnostique de l'interface au réseau sans fil

En cas d'erreur, déclenche une exception ou retourne Y\_MESSAGE\_INVALID.

**wireless→get\_module()**

**YWireless**

**wireless→module()wireless.get\_module()**

---

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( ) As YModule
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

**Retourne :**

une instance de `YModule`

**wireless→get\_security()****YWireless****wireless→security()wireless.get\_security()**

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

```
function get_security( ) As Integer
```

**Retourne :**

une valeur parmi Y\_SECURITY\_UNKNOWN, Y\_SECURITY\_OPEN, Y\_SECURITY\_WEP, Y\_SECURITY\_WPA et Y\_SECURITY\_WPA2 représentant l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y\_SECURITY\_INVALID.

**wireless→get\_ssid()**

**YWireless**

**wireless→ssid()wireless.get\_ssid()**

---

Retourne le nom (SSID) du réseau sans-fil sélectionné.

```
function get_ssid( ) As String
```

**Retourne :**

une chaîne de caractères représentant le nom (SSID) du réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y\_SSID\_INVALID.

**wireless→get(userData)****YWireless****wireless→userData()wireless.get(userData())**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set(userData).

```
function get(userData) As Object
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

**Retourne :**

l'objet stocké précédemment par l'appelant.

**wireless→isOnline()wireless.isOnline()****YWireless**

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

**function isOnline( ) As Boolean**

Si les valeurs des attributs en cache de l'interface réseau sans fil sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

**Retourne :**

true si l'interface réseau sans fil est joignable, false sinon

**wireless→joinNetwork()wireless.joinNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

```
function joinNetwork( ) As Integer
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

**Paramètres :**

**ssid** nom du réseau sans fil à utiliser

**securityKey** clé d'accès au réseau, sous forme de chaîne de caractères

**Retourne :**

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→load()|wireless.load()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
function load( ByVal msValidity As Integer) As YRETCODE
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

**Paramètres :**

**msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

**Retourne :**

YAPI\_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**wireless→nextWireless()wireless.nextWireless()****YWireless**

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

```
function nextWireless( ) As YWireless
```

**Retourne :**

un pointeur sur un objet `YWireless` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**wireless→registerValueCallback()  
wireless.registerValueCallback()****YWireless**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( ) As Integer
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appellés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

**Paramètres :**

**callback** la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**wireless→set\_logicalName()**  
**wireless→setLogicalName()**  
**wireless.set\_logicalName()**

**YWireless**

Modifie le nom logique de l'interface réseau sans fil.

```
function set_logicalName( ByVal newval As String) As Integer
```

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).

```
procedure set(userData( ByVal data As Object)
```

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→softAPNetwork()wireless.softAPNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

function **softAPNetwork( ) As Integer**

Cette fonction ne fonctionne que sur le YoctoHub-Wireless-g.

Si une clef d'accès est configurée pour un réseau SoftAP, le réseau sera protégé par une sécurité WEP40 (5 caractères ou 10 chiffres hexadécimaux) ou WEP128 (13 caractères ou 26 chiffres hexadécimaux). Pour réduire les risques d'intrusion, il est recommandé d'utiliser une clé WEP128 basée sur 26 chiffres hexadécimaux provenant d'une bonne source aléatoire.

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.



# Index

## A

Accelerometer 31  
adhocNetwork, YWireless 1716  
Alimentation 476  
Altitude 72  
AnButton 113

## B

Basic 3  
Blueprint 10  
brakingForceMove, YMotor 852  
Brute 340

## C

calibrate, YLightSensor 717  
calibrateFromPoints, YAccelerometer 35  
calibrateFromPoints, YAltitude 76  
calibrateFromPoints, YCarbonDioxide 154  
calibrateFromPoints, YCompass 220  
calibrateFromPoints, YCurrent 259  
calibrateFromPoints, YGenericSensor 531  
calibrateFromPoints, YGyro 579  
calibrateFromPoints, YHumidity 653  
calibrateFromPoints, YLightSensor 718  
calibrateFromPoints, YMagnetometer 758  
calibrateFromPoints, YPower 970  
calibrateFromPoints, YPressure 1012  
calibrateFromPoints, YPwmInput 1050  
calibrateFromPoints, YQt 1156  
calibrateFromPoints,YSensor 1290  
calibrateFromPoints, YTemperature 1418  
calibrateFromPoints, YTilt 1458  
calibrateFromPoints, YVoc 1496  
calibrateFromPoints, YVoltage 1534  
callbackLogin, YNetwork 893  
cancel3DCalibration, YRefFrame 1220  
CarbonDioxide 150  
checkFirmware, YModule 805  
CheckLogicalName, YAPI 12  
clear, YDisplayLayer 445  
clearConsole, YDisplayLayer 446  
ColorLed 188  
Compass 216  
Configuration 1216  
consoleOut, YDisplayLayer 447  
Contrôle 4, 6, 476, 801, 944  
copyLayerContent, YDisplay 402  
Current 255

## D

DataLogger 293  
delayedPulse, YDigitalIO 359

delayedPulse, YRelay 1255  
delayedPulse, YWatchdog 1673  
describe, YAccelerometer 36  
describe, YAltitude 77  
describe, YAnButton 117  
describe, YCarbonDioxide 155  
describe, YColorLed 191  
describe, YCompass 221  
describe, YCurrent 260  
describe, YDataLogger 297  
describe, YDigitalIO 360  
describe, YDisplay 403  
describe, YDualPower 479  
describe, YFiles 503  
describe, YGenericSensor 532  
describe, YGyro 580  
describe, YHubPort 628  
describe, YHumidity 654  
describe, YLed 690  
describe, YLightSensor 719  
describe, YMagnetometer 759  
describe, YModule 806  
describe, YMotor 853  
describe, YNetwork 894  
describe, YOsControl 947  
describe, YPower 971  
describe, YPressure 1013  
describe, YPwmInput 1051  
describe, YPwmOutput 1097  
describe, YPwmPowerSource 1133  
describe, YQt 1157  
describe, YRealTimeClock 1193  
describe, YRefFrame 1221  
describe, YRelay 1256  
describe, YSensor 1291  
describe, YSerialPort 1329  
describe, YServo 1384  
describe, YTemperature 1419  
describe, YTilt 1459  
describe, YVoc 1497  
describe, YVoltage 1535  
describe, YVSource 1571  
describe, YWakeUpMonitor 1603  
describe, YWakeUpSchedule 1637  
describe, YWatchdog 1674  
describe, YWireless 1717  
DigitalIO 355  
DisableExceptions, YAPI 13  
Display 398  
DisplayLayer 444  
Données 326, 328, 340  
download, YFiles 504  
download, YModule 807  
drawBar, YDisplayLayer 448  
drawBitmap, YDisplayLayer 449

drawCircle, YDisplayLayer 450  
drawDisc, YDisplayLayer 451  
drawImage, YDisplayLayer 452  
drawPixel, YDisplayLayer 453  
drawRect, YDisplayLayer 454  
drawText, YDisplayLayer 455  
drivingForceMove, YMotor 854  
dutyCycleMove, YPwmOutput 1098

## E

EnableExceptions, YAPI 14  
Enregistrées 328, 340  
Erreurs 8

## F

fade, YDisplay 404  
Files 500  
FindAccelerometer, YAccelerometer 33  
FindAltitude, YAltitude 74  
FindAnButton, YAnButton 115  
FindCarbonDioxide, YCarbonDioxide 152  
FindColorLed, YColorLed 189  
FindCompass, YCompass 218  
FindCurrent, YCurrent 257  
FindDataLogger, YDataLogger 295  
FindDigitalIO, YDigitalIO 357  
FindDisplay, YDisplay 400  
FindDualPower, YDualPower 477  
FindFiles, YFiles 501  
FindGenericSensor, YGenericSensor 529  
FindGyro, YGyro 577  
FindHubPort, YHubPort 626  
FindHumidity, YHumidity 651  
FindLed, YLed 688  
FindLightSensor, YLightSensor 715  
FindMagnetometer, YMagnetometer 756  
FindModule, YModule 803  
FindMotor, YMotor 850  
FindNetwork, YNetwork 891  
FindOsControl, YOsControl 945  
FindPower, YPower 968  
FindPressure, YPressure 1010  
FindPwmInput, YPwmInput 1048  
FindPwmOutput, YPwmOutput 1095  
FindPwmPowerSource, YPwmPowerSource 1131  
FindQt, YQt 1154  
FindRealTimeClock, YRealTimeClock 1191  
FindRefFrame, YRefFrame 1218  
FindRelay, YRelay 1253  
FindSensor, YSensor 1288  
FindSerialPort, YSerialPort 1327  
FindServo, YServo 1382  
FindTemperature, YTemperature 1416  
FindTilt, YTilt 1456  
FindVoc, YVoc 1494  
FindVoltage, YVoltage 1532  
FindVSource, YVSource 1569

FindWakeUpMonitor, YWakeUpMonitor 1601  
FindWakeUpSchedule, YWakeUpSchedule 1635  
FindWatchdog, YWatchdog 1671  
FindWireless, YWireless 1714  
FirstAccelerometer, YAccelerometer 34  
FirstAltitude, YAltitude 75  
FirstAnButton, YAnButton 116  
FirstCarbonDioxide, YCarbonDioxide 153  
FirstColorLed, YColorLed 190  
FirstCompass, YCompass 219  
FirstCurrent, YCurrent 258  
FirstDataLogger, YDataLogger 296  
FirstDigitalIO, YDigitalIO 358  
FirstDisplay, YDisplay 401  
FirstDualPower, YDualPower 478  
FirstFiles, YFiles 502  
FirstGenericSensor, YGenericSensor 530  
FirstGyro, YGyro 578  
FirstHubPort, YHubPort 627  
FirstHumidity, YHumidity 652  
FirstLed, YLed 689  
FirstLightSensor, YLightSensor 716  
FirstMagnetometer, YMagnetometer 757  
FirstModule, YModule 804  
FirstMotor, YMotor 851  
FirstNetwork, YNetwork 892  
FirstOsControl, YOsControl 946  
FirstPower, YPower 969  
FirstPressure, YPressure 1011  
FirstPwmInput, YPwmInput 1049  
FirstPwmOutput, YPwmOutput 1096  
FirstPwmPowerSource, YPwmPowerSource 1132  
FirstQt, YQt 1155  
FirstRealTimeClock, YRealTimeClock 1192  
FirstRefFrame, YRefFrame 1219  
FirstRelay, YRelay 1254  
FirstSensor, YSensor 1289  
FirstSerialPort, YSerialPort 1328  
FirstServo, YServo 1383  
FirstTemperature, YTemperature 1417  
FirstTilt, YTilt 1457  
FirstVoc, YVoc 1495  
FirstVoltage, YVoltage 1533  
FirstVSource, YVSource 1570  
FirstWakeUpMonitor, YWakeUpMonitor 1602  
FirstWakeUpSchedule, YWakeUpSchedule 1636  
FirstWatchdog, YWatchdog 1672  
FirstWireless, YWireless 1715  
Fonctions 11, 1286  
forgetAllDataStream, YDataLogger 298  
format\_fs, YFiles 505  
Forme 326  
FreeAPI, YAPI 15  
functionCount, YModule 808  
functionId, YModule 809  
functionName, YModule 810  
functionValue, YModule 811

# G

GenericSensor 527  
get\_3DCalibrationHint, YRefFrame 1222  
get\_3DCalibrationLogMsg, YRefFrame 1223  
get\_3DCalibrationProgress, YRefFrame 1224  
get\_3DCalibrationStage, YRefFrame 1225  
get\_3DCalibrationStageProgress, YRefFrame 1226  
get\_adminPassword, YNetwork 895  
get\_advertisedValue, YAccelerometer 37  
get\_advertisedValue, YAltitude 78  
get\_advertisedValue, YAnButton 118  
get\_advertisedValue, YCarbonDioxide 156  
get\_advertisedValue, YColorLed 192  
get\_advertisedValue, YCompass 222  
get\_advertisedValue, YCurrent 261  
get\_advertisedValue, YDataLogger 299  
get\_advertisedValue, YDigitalIO 361  
get\_advertisedValue, YDisplay 405  
get\_advertisedValue, YDualPower 480  
get\_advertisedValue, YFiles 506  
get\_advertisedValue, YGenericSensor 533  
get\_advertisedValue, YGyro 581  
get\_advertisedValue, YHubPort 629  
get\_advertisedValue, YHumidity 655  
get\_advertisedValue, YLed 691  
get\_advertisedValue, YLightSensor 720  
get\_advertisedValue, YMagnetometer 760  
get\_advertisedValue, YMotor 855  
get\_advertisedValue, YNetwork 896  
get\_advertisedValue, YOsControl 948  
get\_advertisedValue, YPower 972  
get\_advertisedValue, YPressure 1014  
get\_advertisedValue, YPwmInput 1052  
get\_advertisedValue, YPwmOutput 1099  
get\_advertisedValue, YPwmPowerSource 1134  
get\_advertisedValue, YQt 1158  
get\_advertisedValue, YRealTimeClock 1194  
get\_advertisedValue, YRefFrame 1227  
get\_advertisedValue, YRelay 1257  
get\_advertisedValue, YSensor 1292  
get\_advertisedValue, YSerialPort 1331  
get\_advertisedValue, YServo 1385  
get\_advertisedValue, YTemperature 1420  
get\_advertisedValue, YTilt 1460  
get\_advertisedValue, YVoc 1498  
get\_advertisedValue, YVoltage 1536  
get\_advertisedValue, YVSource 1572  
get\_advertisedValue, YWakeUpMonitor 1604  
get\_advertisedValue, YWakeUpSchedule 1638  
get\_advertisedValue, YWatchdog 1675  
get\_advertisedValue, YWireless 1718  
get\_allSettings, YModule 812  
get\_analogCalibration, YAnButton 119  
get\_autoStart, YDataLogger 300  
get\_autoStart, YWatchdog 1676  
get\_averageValue, YDataStream 341  
get\_averageValue, YMeasure 795

get\_baudRate, YHubPort 630  
get\_beacon, YModule 813  
get\_beaconDriven, YDataLogger 301  
get\_bearing, YRefFrame 1228  
get\_bitDirection, YDigitalIO 362  
get\_bitOpenDrain, YDigitalIO 363  
get\_bitPolarity, YDigitalIO 364  
get\_bitState, YDigitalIO 365  
get\_blinking, YLed 692  
get\_brakingForce, YMotor 856  
get\_brightness, YDisplay 406  
get\_calibratedValue, YAnButton 120  
get\_calibrationMax, YAnButton 121  
get\_calibrationMin, YAnButton 122  
get\_callbackCredentials, YNetwork 897  
get\_callbackEncoding, YNetwork 898  
get\_callbackMaxDelay, YNetwork 899  
get\_callbackMethod, YNetwork 900  
get\_callbackMinDelay, YNetwork 901  
get\_callbackUrl, YNetwork 902  
get\_channel, YWireless 1719  
get\_columnCount, YDataStream 342  
get\_columnNames, YDataStream 343  
get\_cosPhi, YPower 973  
get\_countdown, YRelay 1258  
get\_countdown, YWatchdog 1677  
get\_CTS, YSerialPort 1330  
get\_currentRawValue, YAccelerometer 38  
get\_currentRawValue, YAltitude 79  
get\_currentRawValue, YCarbonDioxide 157  
get\_currentRawValue, YCompass 223  
get\_currentRawValue, YCurrent 262  
get\_currentRawValue, YGenericSensor 534  
get\_currentRawValue, YGyro 582  
get\_currentRawValue, YHumidity 656  
get\_currentRawValue, YLightSensor 721  
get\_currentRawValue, YMagnetometer 761  
get\_currentRawValue, YPower 974  
get\_currentRawValue, YPressure 1015  
get\_currentRawValue, YPwmInput 1053  
get\_currentRawValue, YQt 1159  
get\_currentRawValue, YSensor 1293  
get\_currentRawValue, YTemperature 1421  
get\_currentRawValue, YTilt 1461  
get\_currentRawValue, YVoc 1499  
get\_currentRawValue, YVoltage 1537  
get\_currentRunIndex, YDataLogger 302  
get\_currentValue, YAccelerometer 39  
get\_currentValue, YAltitude 80  
get\_currentValue, YCarbonDioxide 158  
get\_currentValue, YCompass 224  
get\_currentValue, YCurrent 263  
get\_currentValue, YGenericSensor 535  
get\_currentValue, YGyro 583  
get\_currentValue, YHumidity 657  
get\_currentValue, YLightSensor 722  
get\_currentValue, YMagnetometer 762  
get\_currentValue, YPower 975  
get\_currentValue, YPressure 1016

get\_currentValue, YPwmInput 1054  
get\_currentValue, YQt 1160  
get\_currentValue, YSensor 1294  
get\_currentValue, YTTemperature 1422  
get\_currentValue, YTilt 1462  
get\_currentValue, YVoc 1500  
get\_currentValue, YVoltage 1538  
get\_cutOffVoltage, YMotor 857  
get\_data, YDataStream 344  
get\_dataRows, YDataStream 345  
get\_dataSamplesIntervalMs, YDataStream 346  
get\_dataSets, YDataLogger 303  
get\_dataStreams, YDataLogger 304  
get\_dateTime, YRealTimeClock 1195  
get\_detectedWlans, YWireless 1720  
get\_discoverable, YNetwork 903  
get\_display, YDisplayLayer 456  
get\_displayHeight, YDisplay 407  
get\_displayHeight, YDisplayLayer 457  
get\_displayLayer, YDisplay 408  
get\_displayType, YDisplay 409  
get\_displayWidth, YDisplay 410  
get\_displayWidth, YDisplayLayer 458  
get\_drivingForce, YMotor 858  
get\_duration, YDataStream 347  
get\_dutyCycle, YPwmInput 1055  
get\_dutyCycle, YPwmOutput 1100  
get\_dutyCycleAtPowerOn, YPwmOutput 1101  
get\_enabled, YDisplay 411  
get\_enabled, YHubPort 631  
get\_enabled, YPwmOutput 1102  
get\_enabled,YServo 1386  
get\_enabledAtPowerOn, YPwmOutput 1103  
get\_enabledAtPowerOn, YServo 1387  
get\_endTimeUTC, YDataSet 329  
get\_endTimeUTC, YMeasure 796  
get\_errCount, YSerialPort 1332  
getErrorMessage, YAccelerometer 40  
getErrorMessage, YAltitude 81  
getErrorMessage, YAnButton 123  
getErrorMessage, YCarbonDioxide 159  
getErrorMessage, YColorLed 193  
getErrorMessage, YCompass 225  
getErrorMessage, YCurrent 264  
getErrorMessage, YDataLogger 305  
getErrorMessage, YDigitalIO 366  
getErrorMessage, YDisplay 412  
getErrorMessage, YDualPower 481  
getErrorMessage, YFiles 507  
getErrorMessage, YGenericSensor 536  
getErrorMessage, YGyro 584  
getErrorMessage, YHubPort 632  
getErrorMessage, YHumidity 658  
getErrorMessage, YLed 693  
getErrorMessage, YLightSensor 723  
getErrorMessage, YMagnetometer 763  
getErrorMessage, YModule 814  
getErrorMessage, YMotor 859  
getErrorMessage, YNetwork 904  
getErrorMessage, YOsControl 949  
getErrorMessage, YPower 976  
getErrorMessage, YPressure 1017  
getErrorMessage, YPwmInput 1056  
getErrorMessage, YPwmOutput 1104  
getErrorMessage, YPwmPowerSource 1135  
getErrorMessage, YQt 1161  
getErrorMessage, YRealTimeClock 1196  
getErrorMessage, YRefFrame 1229  
getErrorMessage, YRelay 1259  
getErrorMessage, YSensor 1295  
getErrorMessage, YSerialPort 1333  
getErrorMessage, YServo 1388  
getErrorMessage, YTTemperature 1423  
getErrorMessage, YTilt 1463  
getErrorMessage, YVoc 1501  
getErrorMessage, YVoltage 1539  
getErrorMessage, YVSource 1573  
getErrorMessage, YWakeUpMonitor 1605  
getErrorMessage, YWakeUpSchedule 1639  
getErrorMessage, YWatchdog 1678  
getErrorMessage, YWireless 1721  
get\_errorType, YAccelerometer 41  
get\_errorType, YAltitude 82  
get\_errorType, YAnButton 124  
get\_errorType, YCarbonDioxide 160  
get\_errorType, YColorLed 194  
get\_errorType, YCompass 226  
get\_errorType, YCurrent 265  
get\_errorType, YDataLogger 306  
get\_errorType, YDigitalIO 367  
get\_errorType, YDisplay 413  
get\_errorType, YDualPower 482  
get\_errorType, YFiles 508  
get\_errorType, YGenericSensor 537  
get\_errorType, YGyro 585  
get\_errorType, YHubPort 633  
get\_errorType, YHumidity 659  
get\_errorType, YLed 694  
get\_errorType, YLightSensor 724  
get\_errorType, YMagnetometer 764  
get\_errorType, YModule 815  
get\_errorType, YMotor 860  
get\_errorType, YNetwork 905  
get\_errorType, YOsControl 950  
get\_errorType, YPower 977  
get\_errorType, YPressure 1018  
get\_errorType, YPwmInput 1057  
get\_errorType, YPwmOutput 1105  
get\_errorType, YPwmPowerSource 1136  
get\_errorType, YQt 1162  
get\_errorType, YRealTimeClock 1197  
get\_errorType, YRefFrame 1230  
get\_errorType, YRelay 1260  
get\_errorType, YSensor 1296  
get\_errorType, YSerialPort 1334  
get\_errorType, YServo 1389  
get\_errorType, YTTemperature 1424  
get\_errorType, YTilt 1464

get\_errorType, YVoc 1502  
get\_errorType, YVoltage 1540  
get\_errorType, YVSource 1574  
get\_errorType, YWakeUpMonitor 1606  
get\_errorType, YWakeUpSchedule 1640  
get\_errorType, YWatchdog 1679  
get\_errorType, YWireless 1722  
get\_extPowerFailure, YVSource 1575  
get\_extVoltage, YDualPower 483  
get\_failSafeTimeout, YMotor 861  
get\_failure, YVSource 1576  
get\_filesCount, YFiles 509  
get\_firmwareRelease, YModule 816  
get\_freeSpace, YFiles 510  
get\_frequency, YMotor 862  
get\_frequency, YPwmInput 1058  
get\_frequency, YPwmOutput 1106  
get\_functionDescriptor, YAccelerometer 42  
get\_functionDescriptor, YAltitude 83  
get\_functionDescriptor, YAnButton 125  
get\_functionDescriptor, YCarbonDioxide 161  
get\_functionDescriptor, YColorLed 195  
get\_functionDescriptor, YCompass 227  
get\_functionDescriptor, YCurrent 266  
get\_functionDescriptor, YDataLogger 307  
get\_functionDescriptor, YDigitalIO 368  
get\_functionDescriptor, YDisplay 414  
get\_functionDescriptor, YDualPower 484  
get\_functionDescriptor, YFiles 511  
get\_functionDescriptor, YGenericSensor 538  
get\_functionDescriptor, YGyro 586  
get\_functionDescriptor, YHubPort 634  
get\_functionDescriptor, YHumidity 660  
get\_functionDescriptor, YLed 695  
get\_functionDescriptor, YLightSensor 725  
get\_functionDescriptor, YMagnetometer 765  
get\_functionDescriptor, YMotor 863  
get\_functionDescriptor, YNetwork 906  
get\_functionDescriptor, YOsControl 951  
get\_functionDescriptor, YPower 978  
get\_functionDescriptor, YPressure 1019  
get\_functionDescriptor, YPwmInput 1059  
get\_functionDescriptor, YPwmOutput 1107  
get\_functionDescriptor, YPwmPowerSource 1137  
get\_functionDescriptor, YQt 1163  
get\_functionDescriptor, YRealTimeClock 1198  
get\_functionDescriptor, YRefFrame 1231  
get\_functionDescriptor, YRelay 1261  
get\_functionDescriptor,YSensor 1297  
get\_functionDescriptor, YSerialPort 1335  
get\_functionDescriptor, YServo 1390  
get\_functionDescriptor, YTilt 1425  
get\_functionDescriptor, YTilt 1465  
get\_functionDescriptor, YVoc 1503  
get\_functionDescriptor, YVoltage 1541  
get\_functionDescriptor, YVSource 1577  
get\_functionDescriptor, YWakeUpMonitor 1607  
get\_functionDescriptor, YWakeUpSchedule 1641  
get\_functionDescriptor, YWatchdog 1680  
get\_functionDescriptor, YWireless 1723  
get\_functionId, YAccelerometer 43  
get\_functionId, YAltitude 84  
get\_functionId, YAnButton 126  
get\_functionId, YCarbonDioxide 162  
get\_functionId, YColorLed 196  
get\_functionId, YCompass 228  
get\_functionId, YCurrent 267  
get\_functionId, YDataLogger 308  
get\_functionId, YDataSet 330  
get\_functionId, YDigitalIO 369  
get\_functionId, YDisplay 415  
get\_functionId, YDualPower 485  
get\_functionId, YFiles 512  
get\_functionId, YGenericSensor 539  
get\_functionId, YGyro 587  
get\_functionId, YHubPort 635  
get\_functionId, YHumidity 661  
get\_functionId, YLed 696  
get\_functionId, YLightSensor 726  
get\_functionId, YMagnetometer 766  
get\_functionId, YMotor 864  
get\_functionId, YNetwork 907  
get\_functionId, YOsControl 952  
get\_functionId, YPower 979  
get\_functionId, YPressure 1020  
get\_functionId, YPwmInput 1060  
get\_functionId, YPwmOutput 1108  
get\_functionId, YPwmPowerSource 1138  
get\_functionId, YQt 1164  
get\_functionId, YRealTimeClock 1199  
get\_functionId, YRefFrame 1232  
get\_functionId, YRelay 1262  
get\_functionId, YSensor 1298  
get\_functionId, YSerialPort 1336  
get\_functionId, YServo 1391  
get\_functionId, YTilt 1426  
get\_functionId, YTilt 1466  
get\_functionId, YVoc 1504  
get\_functionId, YVoltage 1542  
get\_functionId, YVSource 1578  
get\_functionId, YWakeUpMonitor 1608  
get\_functionId, YWakeUpSchedule 1642  
get\_functionId, YWatchdog 1681  
get\_functionId, YWireless 1724  
get\_hardwareId, YAccelerometer 44  
get\_hardwareId, YAltitude 85  
get\_hardwareId, YAnButton 127  
get\_hardwareId, YCarbonDioxide 163  
get\_hardwareId, YColorLed 197  
get\_hardwareId, YCompass 229  
get\_hardwareId, YCurrent 268  
get\_hardwareId, YDataLogger 309  
get\_hardwareId, YDataSet 331  
get\_hardwareId, YDigitalIO 370  
get\_hardwareId, YDisplay 416  
get\_hardwareId, YDualPower 486  
get\_hardwareId, YFiles 513  
get\_hardwareId, YGenericSensor 540

get\_hardwareId, YGyro 588  
get\_hardwareId, YHubPort 636  
get\_hardwareId, YHumidity 662  
get\_hardwareId, YLed 697  
get\_hardwareId, YLightSensor 727  
get\_hardwareId, YMagnetometer 767  
get\_hardwareId, YModule 817  
get\_hardwareId, YMotor 865  
get\_hardwareId, YNetwork 908  
get\_hardwareId, YOsControl 953  
get\_hardwareId, YPower 980  
get\_hardwareId, YPressure 1021  
get\_hardwareId, YPwmInput 1061  
get\_hardwareId, YPwmOutput 1109  
get\_hardwareId, YPwmPowerSource 1139  
get\_hardwareId, YQt 1165  
get\_hardwareId, YRealTimeClock 1200  
get\_hardwareId, YRefFrame 1233  
get\_hardwareId, YRelay 1263  
get\_hardwareId,YSensor 1299  
get\_hardwareId, YSerialPort 1337  
get\_hardwareId, YServo 1392  
get\_hardwareId, YTemperature 1427  
get\_hardwareId, YTilt 1467  
get\_hardwareId, YVoc 1505  
get\_hardwareId, YVoltage 1543  
get\_hardwareId, YVSource 1579  
get\_hardwareId, YWakeUpMonitor 1609  
get\_hardwareId, YWakeUpSchedule 1643  
get\_hardwareId, YWatchdog 1682  
get\_hardwareId, YWireless 1725  
get\_heading, YGyro 589  
get\_highestValue, YAccelerometer 45  
get\_highestValue, YAltitude 86  
get\_highestValue, YCarbonDioxide 164  
get\_highestValue, YCompass 230  
get\_highestValue, YCurrent 269  
get\_highestValue, YGenericSensor 541  
get\_highestValue, YGyro 590  
get\_highestValue, YHumidity 663  
get\_highestValue, YLightSensor 728  
get\_highestValue, YMagnetometer 768  
get\_highestValue, YPower 981  
get\_highestValue, YPressure 1022  
get\_highestValue, YPwmInput 1062  
get\_highestValue, YQt 1166  
get\_highestValue, YSensor 1300  
get\_highestValue, YTemperature 1428  
get\_highestValue, YTilt 1468  
get\_highestValue, YVoc 1506  
get\_highestValue, YVoltage 1544  
get\_hours, YWakeUpSchedule 1644  
get\_hslColor, YColorLed 198  
get\_icon2d, YModule 818  
get\_ipAddress, YNetwork 909  
get\_isPressed, YAnButton 128  
get\_lastLogs, YModule 819  
get\_lastMsg, YSerialPort 1338  
get\_lastTimePressed, YAnButton 129  
get\_lastTimeReleased, YAnButton 130  
get\_layerCount, YDisplay 417  
get\_layerHeight, YDisplay 418  
get\_layerHeight, YDisplayLayer 459  
get\_layerWidth, YDisplay 419  
get\_layerWidth, YDisplayLayer 460  
get\_linkQuality, YWireless 1726  
get\_list, YFiles 514  
get\_logFrequency, YAccelerometer 46  
get\_logFrequency, YAltitude 87  
get\_logFrequency, YCarbonDioxide 165  
get\_logFrequency, YCompass 231  
get\_logFrequency, YCurrent 270  
get\_logFrequency, YGenericSensor 542  
get\_logFrequency, YGyro 591  
get\_logFrequency, YHumidity 664  
get\_logFrequency, YLightSensor 729  
get\_logFrequency, YMagnetometer 769  
get\_logFrequency, YPower 982  
get\_logFrequency, YPressure 1023  
get\_logFrequency, YPwmInput 1063  
get\_logFrequency, YQt 1167  
get\_logFrequency, YSensor 1301  
get\_logFrequency, YTemperature 1429  
get\_logFrequency, YTilt 1469  
get\_logFrequency, YVoc 1507  
get\_logFrequency, YVoltage 1545  
get\_logicalName, YAccelerometer 47  
get\_logicalName, YAltitude 88  
get\_logicalName, YAnButton 131  
get\_logicalName, YCarbonDioxide 166  
get\_logicalName, YColorLed 199  
get\_logicalName, YCompass 232  
get\_logicalName, YCurrent 271  
get\_logicalName, YDataLogger 310  
get\_logicalName, YDigitalIO 371  
get\_logicalName, YDisplay 420  
get\_logicalName, YDualPower 487  
get\_logicalName, YFiles 515  
get\_logicalName, YGenericSensor 543  
get\_logicalName, YGyro 592  
get\_logicalName, YHubPort 637  
get\_logicalName, YHumidity 665  
get\_logicalName, YLed 698  
get\_logicalName, YLightSensor 730  
get\_logicalName, YMagnetometer 770  
get\_logicalName, YModule 820  
get\_logicalName, YMotor 866  
get\_logicalName, YNetwork 910  
get\_logicalName, YOsControl 954  
get\_logicalName, YPower 983  
get\_logicalName, YPressure 1024  
get\_logicalName, YPwmInput 1064  
get\_logicalName, YPwmOutput 1110  
get\_logicalName, YPwmPowerSource 1140  
get\_logicalName, YQt 1168  
get\_logicalName, YRealTimeClock 1201  
get\_logicalName, YRefFrame 1234  
get\_logicalName, YRelay 1264

get\_logicalName, YSensor 1302  
get\_logicalName, YSerialPort 1339  
get\_logicalName,YServo 1393  
get\_logicalName,YTemperature 1430  
get\_logicalName,YTilt 1470  
get\_logicalName,YVoc 1508  
get\_logicalName,YVoltage 1546  
get\_logicalName,YVSource 1580  
get\_logicalName,YWakeUpMonitor 1610  
get\_logicalName,YWakeUpSchedule 1645  
get\_logicalName,YWatchdog 1683  
get\_logicalName,YWireless 1727  
get\_lowestValue, YAccelerometer 48  
get\_lowestValue, YAltitude 89  
get\_lowestValue, YCarbonDioxide 167  
get\_lowestValue, YCompass 233  
get\_lowestValue, YCurrent 272  
get\_lowestValue, YGenericSensor 544  
get\_lowestValue, YGyro 593  
get\_lowestValue, YHumidity 666  
get\_lowestValue, YLightSensor 731  
get\_lowestValue, YMagnetometer 771  
get\_lowestValue, YPower 984  
get\_lowestValue, YPressure 1025  
get\_lowestValue, YPwmInput 1065  
get\_lowestValue, YQt 1169  
get\_lowestValue, YSensor 1303  
get\_lowestValue, YTemperature 1431  
get\_lowestValue, YTilt 1471  
get\_lowestValue, YVoc 1509  
get\_lowestValue, YVoltage 1547  
get\_luminosity, YLed 699  
get\_luminosity, YModule 821  
get\_macAddress, YNetwork 911  
get\_magneticHeading, YCompass 234  
get\_maxTimeOnStateA, YRelay 1265  
get\_maxTimeOnStateA, YWatchdog 1684  
get\_maxTimeOnStateB, YRelay 1266  
get\_maxTimeOnStateB, YWatchdog 1685  
get\_maxValue, YDataStream 348  
get\_maxValue, YMeasure 797  
get\_measures, YDataSet 332  
get\_measureType, YLightSensor 732  
get\_message, YWireless 1728  
get\_meter, YPower 985  
get\_meterTimer, YPower 986  
get\_minutes, YWakeUpSchedule 1646  
get\_minutesA, YWakeUpSchedule 1647  
get\_minutesB, YWakeUpSchedule 1648  
get\_minValue, YDataStream 349  
get\_minValue, YMeasure 798  
get\_module, YAccelerometer 49  
get\_module, YAltitude 90  
get\_module, YAnButton 132  
get\_module, YCarbonDioxide 168  
get\_module, YColorLed 200  
get\_module, YCompass 235  
get\_module, YCurrent 273  
get\_module, YDataLogger 311  
get\_module, YDigitalIO 372  
get\_module, YDisplay 421  
get\_module, YDualPower 488  
get\_module, YFiles 516  
get\_module, YGenericSensor 545  
get\_module, YGyro 594  
get\_module, YHubPort 638  
get\_module, YHumidity 667  
get\_module, YLed 700  
get\_module, YLightSensor 733  
get\_module, YMagnetometer 772  
get\_module, YMotor 867  
get\_module, YNetwork 912  
get\_module, YOsControl 955  
get\_module, YPower 987  
get\_module, YPressure 1026  
get\_module, YPwmInput 1066  
get\_module, YPwmOutput 1111  
get\_module, YPwmPowerSource 1141  
get\_module, YQt 1170  
get\_module, YRealTimeClock 1202  
get\_module, YRefFrame 1235  
get\_module, YRelay 1267  
get\_module, YSensor 1304  
get\_module, YSerialPort 1340  
get\_module, YServo 1394  
get\_module, YTemperature 1432  
get\_module, YTilt 1472  
get\_module, YVoc 1510  
get\_module, YVoltage 1548  
get\_module, YVSource 1581  
get\_module, YWakeUpMonitor 1611  
get\_module, YWakeUpSchedule 1649  
get\_module, YWatchdog 1686  
get\_module, YWireless 1729  
get\_monthDays, YWakeUpSchedule 1650  
get\_months, YWakeUpSchedule 1651  
get\_motorStatus, YMotor 868  
get\_mountOrientation, YRefFrame 1236  
get\_mountPosition, YRefFrame 1237  
get\_msgCount, YSerialPort 1341  
get\_neutral, YServo 1395  
get\_nextOccurrence, YWakeUpSchedule 1652  
get\_nextWakeUp, YWakeUpMonitor 1612  
get\_orientation, YDisplay 422  
get\_output, YRelay 1268  
get\_output, YWatchdog 1687  
get\_outputVoltage, YDigitalIO 373  
get\_overCurrent, YVSource 1582  
get\_overCurrentLimit, YMotor 869  
get\_overHeat, YVSource 1583  
get\_overLoad, YVSource 1584  
get\_period, YPwmInput 1067  
get\_period, YPwmOutput 1112  
get\_persistentSettings, YModule 822  
get\_pitch, YGyro 595  
get\_poeCurrent, YNetwork 913  
get\_portDirection, YDigitalIO 374  
get\_portOpenDrain, YDigitalIO 375

get\_portPolarity, YDigitalIO 376  
get\_portSize, YDigitalIO 377  
get\_portState, YDigitalIO 378  
get\_portState, YHubPort 639  
get\_position,YServo 1396  
get\_positionAtPowerOn,YServo 1397  
get\_power,YLed 701  
get\_powerControl,YDualPower 489  
get\_powerDuration,YWakeUpMonitor 1613  
get\_powerMode,YPwmPowerSource 1142  
get\_powerState,YDualPower 490  
get\_preview,YDataSet 333  
get\_primaryDNS,YNetwork 914  
get\_productId,YModule 823  
get\_productName,YModule 824  
get\_productRelease,YModule 825  
get\_progress,YDataSet 334  
get\_protocol,YSerialPort 1342  
get\_pulseCounter,YAnButton 133  
get\_pulseCounter,YPwmInput 1068  
get\_pulseDuration,YPwmInput 1069  
get\_pulseDuration,YPwmOutput 1113  
get\_pulseTimer,YAnButton 134  
get\_pulseTimer,YPwmInput 1070  
get\_pulseTimer,YRelay 1269  
get\_pulseTimer,YWatchdog 1688  
get\_pwmReportMode,YPwmInput 1071  
get\_qnh,YAltitude 91  
get\_quaternionW,YGyro 596  
get\_quaternionX,YGyro 597  
get\_quaternionY,YGyro 598  
get\_quaternionZ,YGyro 599  
get\_range,YServo 1398  
get\_rawValue,YAnButton 135  
get\_readiness,YNetwork 915  
get\_rebootCountdown,YModule 826  
get\_recordedData,YAccelerometer 50  
get\_recordedData,YAltitude 92  
get\_recordedData,YCarbonDioxide 169  
get\_recordedData,YCompass 236  
get\_recordedData,YCurrent 274  
get\_recordedData,YGenericSensor 546  
get\_recordedData,YGyro 600  
get\_recordedData,YHumidity 668  
get\_recordedData,YLightSensor 734  
get\_recordedData,YMagnetometer 773  
get\_recordedData,YPower 988  
get\_recordedData,YPressure 1027  
get\_recordedData,YPwmInput 1072  
get\_recordedData,YQt 1171  
get\_recordedData,YSensor 1305  
get\_recordedData,YTemperature 1433  
get\_recordedData,YTilt 1473  
get\_recordedData,YVoc 1511  
get\_recordedData,YVoltage 1549  
get\_recording,YDataLogger 312  
get\_regulationFailure,YVSource 1585  
get\_reportFrequency,YAccelerometer 51  
get\_reportFrequency,YAltitude 93  
get\_reportFrequency,YCarbonDioxide 170  
get\_reportFrequency,YCompass 237  
get\_reportFrequency,YCurrent 275  
get\_reportFrequency,YGenericSensor 547  
get\_reportFrequency,YGyro 601  
get\_reportFrequency,YHumidity 669  
get\_reportFrequency,YLightSensor 735  
get\_reportFrequency,YMagnetometer 774  
get\_reportFrequency,YPower 989  
get\_reportFrequency,YPressure 1028  
get\_reportFrequency,YPwmInput 1073  
get\_reportFrequency,YQt 1172  
get\_reportFrequency,YSensor 1306  
get\_reportFrequency,YTemperature 1434  
get\_reportFrequency,YTilt 1474  
get\_reportFrequency,YVoc 1512  
get\_reportFrequency,YVoltage 1550  
get\_resolution,YAccelerometer 52  
get\_resolution,YAltitude 94  
get\_resolution,YCarbonDioxide 171  
get\_resolution,YCompass 238  
get\_resolution,YCurrent 276  
get\_resolution,YGenericSensor 548  
get\_resolution,YGyro 602  
get\_resolution,YHumidity 670  
get\_resolution,YLightSensor 736  
get\_resolution,YMagnetometer 775  
get\_resolution,YPower 990  
get\_resolution,YPressure 1029  
get\_resolution,YPwmInput 1074  
get\_resolution,YQt 1173  
get\_resolution,YSensor 1307  
get\_resolution,YTemperature 1435  
get\_resolution,YTilt 1475  
get\_resolution,YVoc 1513  
get\_resolution,YVoltage 1551  
get\_rgbColor,YColorLed 201  
get\_rgbColorAtPowerOn,YColorLed 202  
get\_roll,YGyro 603  
get\_router,YNetwork 916  
getRowCount,YDataStream 350  
get\_runIndex,YDataStream 351  
get\_running,YWatchdog 1689  
get\_rxCount,YSerialPort 1343  
get\_secondaryDNS,YNetwork 917  
get\_security,YWireless 1730  
get\_sensitivity,YAnButton 136  
get\_sensorType,YTemperature 1436  
get\_serialMode,YSerialPort 1344  
get\_serialNumber,YModule 827  
get\_shutdownCountdown,YOsControl 956  
get\_signalBias,YGenericSensor 549  
get\_signalRange,YGenericSensor 550  
get\_signalUnit,YGenericSensor 551  
get\_signalValue,YGenericSensor 552  
get\_sleepCountdown,YWakeUpMonitor 1614  
get\_ssId,YWireless 1731  
get\_starterTime,YMotor 870  
get\_startTime,YDataStream 352

get\_startTimeUTC, YDataRun 326  
get\_startTimeUTC, YDataSet 335  
get\_startTimeUTC, YDataStream 353  
get\_startTimeUTC, YMeasure 799  
get\_startupSeq, YDisplay 423  
get\_state, YRelay 1270  
get\_state, YWatchdog 1690  
get\_stateAtPowerOn, YRelay 1271  
get\_stateAtPowerOn, YWatchdog 1691  
get\_subnetMask, YNetwork 918  
get\_summary, YDataSet 336  
get\_timeSet, YRealTimeClock 1203  
get\_timeUTC, YDataLogger 313  
get\_triggerDelay, YWatchdog 1692  
get\_triggerDuration, YWatchdog 1693  
get\_txCount, YSerialPort 1345  
get\_unit, YAccelerometer 53  
get\_unit, YAltitude 95  
get\_unit, YCarbonDioxide 172  
get\_unit, YCompass 239  
get\_unit, YCurrent 277  
get\_unit, YDataSet 337  
get\_unit, YGenericSensor 553  
get\_unit, YGyro 604  
get\_unit, YHumidity 671  
get\_unit, YLightSensor 737  
get\_unit, YMagnetometer 776  
get\_unit, YPower 991  
get\_unit, YPressure 1030  
get\_unit, YPwmInput 1075  
get\_unit, YQt 1174  
get\_unit,YSensor 1308  
get\_unit, YTTemperature 1437  
get\_unit, YTilt 1476  
get\_unit, YVoc 1514  
get\_unit, YVoltage 1552  
get\_unit, YVSource 1586  
get\_unixTime, YRealTimeClock 1204  
get\_upTime, YModule 828  
get\_usbCurrent, YModule 829  
get\_userData, YAccelerometer 54  
get\_userData, YAltitude 96  
get\_userData, YAnButton 137  
get\_userData, YCarbonDioxide 173  
get\_userData, YColorLed 203  
get\_userData, YCompass 240  
get\_userData, YCurrent 278  
get\_userData, YDataLogger 314  
get\_userData, YDigitalIO 379  
get\_userData, YDisplay 424  
get\_userData, YDualPower 491  
get\_userData, YFiles 517  
get\_userData, YGenericSensor 554  
get\_userData, YGyro 605  
get\_userData, YHubPort 640  
get\_userData, YHumidity 672  
get\_userData, YLed 702  
get\_userData, YLightSensor 738  
get\_userData, YMagnetometer 777

get\_userData, YModule 830  
get\_userData, YMotor 871  
get\_userData, YNetwork 919  
get\_userData, YOsControl 957  
get\_userData, YPower 992  
get\_userData, YPressure 1031  
get\_userData, YPwmInput 1076  
get\_userData, YPwmOutput 1114  
get\_userData, YPwmPowerSource 1143  
get\_userData, YQt 1175  
get\_userData, YRealTimeClock 1205  
get\_userData, YRefFrame 1238  
get\_userData, YRelay 1272  
get\_userData, YSensor 1309  
get\_userData, YSerialPort 1346  
get\_userData, YServo 1399  
get\_userData, YTTemperature 1438  
get\_userData, YTilt 1477  
get\_userData, YVoc 1515  
get\_userData, YVoltage 1553  
get\_userData, YVSource 1587  
get\_userData, YWakeUpMonitor 1615  
get\_userData, YWakeUpSchedule 1653  
get\_userData, YWatchdog 1694  
get\_userData, YWireless 1732  
get\_userPassword, YNetwork 920  
get\_userVar, YModule 831  
get\_utcOffset, YRealTimeClock 1206  
get\_valueRange, YGenericSensor 555  
get\_voltage, YVSource 1588  
get\_wakeUpReason, YWakeUpMonitor 1616  
get\_wakeUpState, YWakeUpMonitor 1617  
get\_weekDays, YWakeUpSchedule 1654  
get\_wwwWatchdogDelay, YNetwork 921  
get\_xValue, YAccelerometer 55  
get\_xValue, YGyro 606  
get\_xValue, YMagnetometer 778  
get\_yValue, YAccelerometer 56  
get\_yValue, YGyro 607  
get\_yValue, YMagnetometer 779  
get\_zValue, YAccelerometer 57  
get\_zValue, YGyro 608  
get\_zValue, YMagnetometer 780  
GetAPIVersion, YAPI 16  
GetTickCount, YAPI 17  
Gyro 575

## H

HandleEvents, YAPI 18  
hide, YDisplayLayer 461  
Horloge 1190  
hslMove, YColorLed 204  
Humidity 649

## I

InitAPI, YAPI 19  
Installation 3  
Interface 31, 72, 113, 150, 188, 216, 255, 293,

355, 398, 444, 476, 500, 527, 575, 625, 649, 687, 713, 754, 801, 848, 888, 966, 1008, 1046, 1093, 1130, 1152, 1190, 1251, 1286, 1324, 1380, 1414, 1454, 1492, 1530, 1568, 1599, 1633, 1669, 1713

## Introduction 1

isOnline, YAccelerometer 58  
isOnline, YAltitude 97  
isOnline, YAnButton 138  
isOnline, YCarbonDioxide 174  
isOnline, YColorLed 205  
isOnline, YCompass 241  
isOnline, YCurrent 279  
isOnline, YDataLogger 315  
isOnline, YDigitalIO 380  
isOnline, YDisplay 425  
isOnline, YDualPower 492  
isOnline, YFiles 518  
isOnline, YGenericSensor 556  
isOnline, YGyro 609  
isOnline, YHubPort 641  
isOnline, YHumidity 673  
isOnline, YLed 703  
isOnline, YLightSensor 739  
isOnline, YMagnetometer 781  
isOnline, YModule 832  
isOnline, YMotor 872  
isOnline, YNetwork 922  
isOnline, YOsControl 958  
isOnline, YPower 993  
isOnline, YPressure 1032  
isOnline, YPwmInput 1077  
isOnline, YPwmOutput 1115  
isOnline, YPwmPowerSource 1144  
isOnline, YQt 1176  
isOnline, YRealTimeClock 1207  
isOnline, YRefFrame 1239  
isOnline, YRelay 1273  
isOnline,YSensor 1310  
isOnline, YSerialPort 1347  
isOnline, YServo 1400  
isOnline, YTemperature 1439  
isOnline, YTilt 1478  
isOnline, YVoc 1516  
isOnline, YVoltage 1554  
isOnline, YVSource 1589  
isOnline, YWakeUpMonitor 1618  
isOnline, YWakeUpSchedule 1655  
isOnline, YWatchdog 1695  
isOnline, YWireless 1733

## J

joinNetwork, YWireless 1734

## K

keepALive, YMotor 873

## L

LightSensor 713  
lineTo, YDisplayLayer 462  
load, YAccelerometer 59  
load, YAltitude 98  
load, YAnButton 139  
load, YCarbonDioxide 175  
load, YColorLed 206  
load, YCompass 242  
load, YCurrent 280  
load, YDataLogger 316  
load, YDigitalIO 381  
load, YDisplay 426  
load, YDualPower 493  
load, YFiles 519  
load, YGenericSensor 557  
load, YGyro 610  
load, YHubPort 642  
load, YHumidity 674  
load, YLed 704  
load, YLightSensor 740  
load, YMagnetometer 782  
load, YModule 833  
load, YMotor 874  
load, YNetwork 923  
load, YOsControl 959  
load, YPower 994  
load, YPressure 1033  
load, YPwmInput 1078  
load, YPwmOutput 1116  
load, YPwmPowerSource 1145  
load, YQt 1177  
load, YRealTimeClock 1208  
load, YRefFrame 1240  
load, YRelay 1274  
load, YSensor 1311  
load, YSerialPort 1348  
load, YServo 1401  
load, YTemperature 1440  
load, YTilt 1479  
load, YVoc 1517  
load, YVoltage 1555  
load, YVSource 1590  
load, YWakeUpMonitor 1619  
load, YWakeUpSchedule 1656  
load, YWatchdog 1696  
load, YWireless 1735  
loadCalibrationPoints, YAccelerometer 60  
loadCalibrationPoints, YAltitude 99  
loadCalibrationPoints, YCarbonDioxide 176  
loadCalibrationPoints, YCompass 243  
loadCalibrationPoints, YCurrent 281  
loadCalibrationPoints, YGenericSensor 558  
loadCalibrationPoints, YGyro 611  
loadCalibrationPoints, YHumidity 675  
loadCalibrationPoints, YLightSensor 741  
loadCalibrationPoints, YMagnetometer 783  
loadCalibrationPoints, YPower 995

loadCalibrationPoints, YPressure 1034  
loadCalibrationPoints, YPwmInput 1079  
loadCalibrationPoints, YQt 1178  
loadCalibrationPoints, YSensor 1312  
loadCalibrationPoints, YTemperature 1441  
loadCalibrationPoints, YTilt 1480  
loadCalibrationPoints, YVoc 1518  
loadCalibrationPoints, YVoltage 1556  
loadMore, YDataSet 338

## M

Magnetometer 754  
Mesurée 795  
Mise 326  
modbusReadBits, YSerialPort 1349  
modbusReadInputBits, YSerialPort 1350  
modbusReadInputRegisters, YSerialPort 1351  
modbusReadRegisters, YSerialPort 1352  
modbusWriteAndReadRegisters, YSerialPort 1353  
modbusWriteBit, YSerialPort 1354  
modbusWriteBits, YSerialPort 1355  
modbusWriteRegister, YSerialPort 1356  
modbusWriteRegisters, YSerialPort 1357  
Module 6, 801  
more3DCalibration, YRefFrame 1241  
Motor 848  
move,YServo 1402  
moveTo, YDisplayLayer 463

## N

.NET 3  
Network 888  
newSequence, YDisplay 427  
nextAccelerometer, YAcelerometer 61  
nextAltitude, YAltitude 100  
nextAnButton, YAnButton 140  
nextCarbonDioxide, YCarbonDioxide 177  
nextColorLed, YColorLed 207  
nextCompass, YCompass 244  
nextCurrent, YCurrent 282  
nextDataLogger, YDataLogger 317  
nextDigitalIO, YDigitalIO 382  
nextDisplay, YDisplay 428  
nextDualPower, YDualPower 494  
nextFiles, YFiles 520  
nextGenericSensor, YGenericSensor 559  
nextGyro, YGyro 612  
nextHubPort, YHubPort 643  
nextHumidity, YHumidity 676  
nextLed, YLed 705  
nextLightSensor, YLightSensor 742  
nextMagnetometer, YMagnetometer 784  
nextModule, YModule 834  
nextMotor, YMotor 875  
nextNetwork, YNetwork 924  
nextOsControl, YOsControl 960  
nextPower, YPower 996

nextPressure, YPressure 1035  
nextPwmInput, YPwmInput 1080  
nextPwmOutput, YPwmOutput 1117  
nextPwmPowerSource, YPwmPowerSource 1146  
nextQt, YQt 1179  
nextRealTimeClock, YRealTimeClock 1209  
nextRefFrame, YRefFrame 1242  
nextRelay, YRelay 1275  
nextSensor, YSensor 1313  
nextSerialPort, YSerialPort 1358  
nextServo, YServo 1403  
nextTemperature, YTemperature 1442  
nextTilt, YTilt 1481  
nextVoc, YVoc 1519  
nextVoltage, YVoltage 1557  
nextVSource, YVSource 1591  
nextWakeUpMonitor, YWakeUpMonitor 1620  
nextWakeUpSchedule, YWakeUpSchedule 1657  
nextWatchdog, YWatchdog 1697  
nextWireless, YWireless 1736

## O

Objets 444

## P

pauseSequence, YDisplay 429  
ping, YNetwork 925  
playSequence, YDisplay 430  
Port 625  
Power 966  
PreregisterHub, YAPI 20  
Pressure 1008  
Projet 3  
pulse, YDigitalIO 383  
pulse, YRelay 1276  
pulse, YVSource 1592  
pulse, YWatchdog 1698  
pulseDurationMove, YPwmOutput 1118  
PwmInput 1046  
PwmPowerSource 1130

## Q

Quaternion 1152  
queryLine, YSerialPort 1359  
queryMODBUS, YSerialPort 1360

## R

read\_seek, YSerialPort 1365  
readHex, YSerialPort 1361  
readLine, YSerialPort 1362  
readMessages, YSerialPort 1363  
readStr, YSerialPort 1364  
Real 1190  
reboot, YModule 835  
Reference 10  
Référentiel 1216

registerAnglesCallback, YGyro 613  
RegisterDeviceArrivalCallback, YAPI 21  
RegisterDeviceRemovalCallback, YAPI 22  
RegisterHub, YAPI 23  
RegisterHubDiscoveryCallback, YAPI 24  
registerLogCallback, YModule 836  
RegisterLogFunction, YAPI 25  
registerQuaternionCallback, YGyro 614  
registerTimedReportCallback, YAccelerometer 62  
registerTimedReportCallback, YAltitude 101  
registerTimedReportCallback, YCarbonDioxide 178  
registerTimedReportCallback, YCompass 245  
registerTimedReportCallback, YCurrent 283  
registerTimedReportCallback, YGenericSensor 560  
registerTimedReportCallback, YGyro 615  
registerTimedReportCallback, YHumidity 677  
registerTimedReportCallback, YLightSensor 743  
registerTimedReportCallback, YMagnetometer 785  
registerTimedReportCallback, YPower 997  
registerTimedReportCallback, YPressure 1036  
registerTimedReportCallback, YPwmInput 1081  
registerTimedReportCallback, YQt 1180  
registerTimedReportCallback,YSensor 1314  
registerTimedReportCallback, YTilt 1443  
registerTimedReportCallback, YTilt 1482  
registerTimedReportCallback, YVoc 1520  
registerTimedReportCallback, YVoltage 1558  
registerValueCallback, YAccelerometer 63  
registerValueCallback, YAltitude 102  
registerValueCallback, YAnButton 141  
registerValueCallback, YCarbonDioxide 179  
registerValueCallback, YColorLed 208  
registerValueCallback, YCompass 246  
registerValueCallback, YCurrent 284  
registerValueCallback, YDataLogger 318  
registerValueCallback, YDigitalIO 384  
registerValueCallback, YDisplay 431  
registerValueCallback, YDualPower 495  
registerValueCallback, YFiles 521  
registerValueCallback, YGenericSensor 561  
registerValueCallback, YGyro 616  
registerValueCallback, YHubPort 644  
registerValueCallback, YHumidity 678  
registerValueCallback, YLed 706  
registerValueCallback, YLightSensor 744  
registerValueCallback, YMagnetometer 786  
registerValueCallback, YMotor 876  
registerValueCallback, YNetwork 926  
registerValueCallback, YOsControl 961  
registerValueCallback, YPower 998  
registerValueCallback, YPressure 1037  
registerValueCallback, YPwmInput 1082  
registerValueCallback, YPwmOutput 1119  
registerValueCallback, YPwmPowerSource 1147  
registerValueCallback, YQt 1181  
registerValueCallback, YRealTimeClock 1210  
registerValueCallback, YRefFrame 1243  
registerValueCallback, YRelay 1277  
registerValueCallback, YSensor 1315  
registerValueCallback, YSerialPort 1366  
registerValueCallback, YServo 1404  
registerValueCallback, YTilt 1444  
registerValueCallback, YTilt 1483  
registerValueCallback, YVoc 1521  
registerValueCallback, YVoltage 1559  
registerValueCallback, YVSource 1593  
registerValueCallback, YWakeUpMonitor 1621  
registerValueCallback, YWakeUpSchedule 1658  
registerValueCallback, YWatchdog 1699  
registerValueCallback, YWireless 1737  
Relay 1251  
remove, YFiles 522  
reset, YDisplayLayer 464  
reset, YPower 999  
reset, YSerialPort 1367  
resetAll, YDisplay 432  
resetCounter, YAnButton 142  
resetCounter, YPwmInput 1083  
resetSleepCountDown, YWakeUpMonitor 1622  
resetStatus, YMotor 877  
resetWatchdog, YWatchdog 1700  
revertFromFlash, YModule 837  
rgbMove, YColorLed 209

## S

save3DCalibration, YRefFrame 1244  
saveSequence, YDisplay 433  
saveToFlash, YModule 838  
selectColorPen, YDisplayLayer 465  
selectEraser, YDisplayLayer 466  
selectFont, YDisplayLayer 467  
selectGrayPen, YDisplayLayer 468  
Senseur 1286  
Séquence 326, 328, 340  
SerialPort 1324  
Servo 1380  
set\_adminPassword, YNetwork 927  
set\_allSettings, YModule 839  
set\_analogCalibration, YAnButton 143  
set\_autoStart, YDataLogger 319  
set\_autoStart, YWatchdog 1701  
set\_beacon, YModule 840  
set\_beaconDriven, YDataLogger 320  
set\_bearing, YRefFrame 1245  
set\_bitDirection, YDigitalIO 385  
set\_bitOpenDrain, YDigitalIO 386  
set\_bitPolarity, YDigitalIO 387  
set\_bitState, YDigitalIO 388  
set\_blinking, YLed 707  
set\_brakingForce, YMotor 878  
set\_brightness, YDisplay 434  
set\_calibrationMax, YAnButton 144  
set\_calibrationMin, YAnButton 145

set\_callbackCredentials, YNetwork 928  
set\_callbackEncoding, YNetwork 929  
set\_callbackMaxDelay, YNetwork 930  
set\_callbackMethod, YNetwork 931  
set\_callbackMinDelay, YNetwork 932  
set\_callbackUrl, YNetwork 933  
set\_currentValue, YAltitude 103  
set\_cutOffVoltage, YMotor 879  
set\_discoverable, YNetwork 934  
set\_drivingForce, YMotor 880  
set\_dutyCycle, YPwmOutput 1120  
set\_dutyCycleAtPowerOn, YPwmOutput 1121  
set\_enabled, YDisplay 435  
set\_enabled, YHubPort 645  
set\_enabled, YPwmOutput 1122  
set\_enabled,YServo 1405  
set\_enabledAtPowerOn, YPwmOutput 1123  
set\_enabledAtPowerOn,YServo 1406  
set\_failSafeTimeout, YMotor 881  
set\_frequency, YMotor 882  
set\_frequency, YPwmOutput 1124  
set\_highestValue, YAccelerometer 64  
set\_highestValue, YAltitude 104  
set\_highestValue, YCarbonDioxide 180  
set\_highestValue, YCompass 247  
set\_highestValue, YCurrent 285  
set\_highestValue, YGenericSensor 562  
set\_highestValue, YGyro 617  
set\_highestValue, YHumidity 679  
set\_highestValue, YLightSensor 745  
set\_highestValue, YMagnetometer 787  
set\_highestValue, YPower 1000  
set\_highestValue, YPressure 1038  
set\_highestValue, YPwmInput 1084  
set\_highestValue, YQt 1182  
set\_highestValue, YSensor 1316  
set\_highestValue, YTemperature 1445  
set\_highestValue, YTilt 1484  
set\_highestValue, YVoc 1522  
set\_highestValue, YVoltage 1560  
set\_hours, YWakeUpSchedule 1659  
set\_hslColor, YColorLed 210  
set\_logFrequency, YAccelerometer 65  
set\_logFrequency, YAltitude 105  
set\_logFrequency, YCarbonDioxide 181  
set\_logFrequency, YCompass 248  
set\_logFrequency, YCurrent 286  
set\_logFrequency, YGenericSensor 563  
set\_logFrequency, YGyro 618  
set\_logFrequency, YHumidity 680  
set\_logFrequency, YLightSensor 746  
set\_logFrequency, YMagnetometer 788  
set\_logFrequency, YPower 1001  
set\_logFrequency, YPressure 1039  
set\_logFrequency, YPwmInput 1085  
set\_logFrequency, YQt 1183  
set\_logFrequency, YSensor 1317  
set\_logFrequency, YTemperature 1446  
set\_logFrequency, YTilt 1485  
set\_logFrequency, YVoc 1523  
set\_logFrequency, YVoltage 1561  
set\_logicalName, YAccelerometer 66  
set\_logicalName, YAltitude 106  
set\_logicalName, YAnButton 146  
set\_logicalName, YCarbonDioxide 182  
set\_logicalName, YColorLed 211  
set\_logicalName, YCompass 249  
set\_logicalName, YCurrent 287  
set\_logicalName, YDataLogger 321  
set\_logicalName, YDigitalIO 389  
set\_logicalName, YDisplay 436  
set\_logicalName, YDualPower 496  
set\_logicalName, YFiles 523  
set\_logicalName, YGenericSensor 564  
set\_logicalName, YGyro 619  
set\_logicalName, YHubPort 646  
set\_logicalName, YHumidity 681  
set\_logicalName, YLed 708  
set\_logicalName, YLightSensor 747  
set\_logicalName, YMagnetometer 789  
set\_logicalName, YModule 841  
set\_logicalName, YMotor 883  
set\_logicalName, YNetwork 935  
set\_logicalName, YOsControl 962  
set\_logicalName, YPower 1002  
set\_logicalName, YPressure 1040  
set\_logicalName, YPwmInput 1086  
set\_logicalName, YPwmOutput 1125  
set\_logicalName, YPwmPowerSource 1148  
set\_logicalName, YQt 1184  
set\_logicalName, YRealTimeClock 1211  
set\_logicalName, YRefFrame 1246  
set\_logicalName, YRelay 1278  
set\_logicalName, YSensor 1318  
set\_logicalName, YSerialPort 1369  
set\_logicalName, YServo 1407  
set\_logicalName, YTemperature 1447  
set\_logicalName, YTilt 1486  
set\_logicalName, YVoc 1524  
set\_logicalName, YVoltage 1562  
set\_logicalName, YVSource 1594  
set\_logicalName, YWakeUpMonitor 1623  
set\_logicalName, YWakeUpSchedule 1660  
set\_logicalName, YWatchdog 1702  
set\_logicalName, YWireless 1738  
set\_lowestValue, YAccelerometer 67  
set\_lowestValue, YAltitude 107  
set\_lowestValue, YCarbonDioxide 183  
set\_lowestValue, YCompass 250  
set\_lowestValue, YCurrent 288  
set\_lowestValue, YGenericSensor 565  
set\_lowestValue, YGyro 620  
set\_lowestValue, YHumidity 682  
set\_lowestValue, YLightSensor 748  
set\_lowestValue, YMagnetometer 790  
set\_lowestValue, YPower 1003  
set\_lowestValue, YPressure 1041  
set\_lowestValue, YPwmInput 1087

set\_lowestValue, YQt 1185  
set\_lowestValue, YSensor 1319  
set\_lowestValue, YTemperature 1448  
set\_lowestValue, YTilt 1487  
set\_lowestValue, YVoc 1525  
set\_lowestValue, YVoltage 1563  
set\_luminosity, YLed 709  
set\_luminosity, YModule 842  
set\_maxTimeOnStateA, YRelay 1279  
set\_maxTimeOnStateA, YWatchdog 1703  
set\_maxTimeOnStateB, YRelay 1280  
set\_maxTimeOnStateB, YWatchdog 1704  
set\_measureType, YLightSensor 749  
set\_minutes, YWakeUpSchedule 1661  
set\_minutesA, YWakeUpSchedule 1662  
set\_minutesB, YWakeUpSchedule 1663  
set\_monthDays, YWakeUpSchedule 1664  
set\_months, YWakeUpSchedule 1665  
set\_mountPosition, YRefFrame 1247  
set\_neutral,YServo 1408  
set\_nextWakeUp, YWakeUpMonitor 1624  
set\_orientation, YDisplay 437  
set\_output, YRelay 1281  
set\_output, YWatchdog 1705  
set\_outputVoltage, YDigitalIO 390  
set\_overCurrentLimit, YMotor 884  
set\_period, YPwmOutput 1126  
set\_portDirection, YDigitalIO 391  
set\_portOpenDrain, YDigitalIO 392  
set\_portPolarity, YDigitalIO 393  
set\_portState, YDigitalIO 394  
set\_position, YServo 1409  
set\_positionAtPowerOn, YServo 1410  
set\_power, YLed 710  
set\_powerControl, YDualPower 497  
set\_powerDuration, YWakeUpMonitor 1625  
set\_powerMode, YPwmPowerSource 1149  
set\_primaryDNS, YNetwork 936  
set\_protocol, YSerialPort 1370  
set\_pulseDuration, YPwmOutput 1127  
set\_pwmReportMode, YPwmInput 1088  
set\_qnh, YAltitude 108  
set\_range, YServo 1411  
set\_recording, YDataLogger 322  
set\_reportFrequency, YAccelerometer 68  
set\_reportFrequency, YAltitude 109  
set\_reportFrequency, YCarbonDioxide 184  
set\_reportFrequency, YCompass 251  
set\_reportFrequency, YCurrent 289  
set\_reportFrequency, YGenericSensor 566  
set\_reportFrequency, YGyro 621  
set\_reportFrequency, YHumidity 683  
set\_reportFrequency, YLightSensor 750  
set\_reportFrequency, YMagnetometer 791  
set\_reportFrequency, YPower 1004  
set\_reportFrequency, YPressure 1042  
set\_reportFrequency, YPwmInput 1089  
set\_reportFrequency, YQt 1186  
set\_reportFrequency, YSensor 1320  
set\_reportFrequency, YTemperature 1449  
set\_reportFrequency, YTilt 1488  
set\_reportFrequency, YVoc 1526  
set\_reportFrequency, YVoltage 1564  
set\_resolution, YAccelerometer 69  
set\_resolution, YAltitude 110  
set\_resolution, YCarbonDioxide 185  
set\_resolution, YCompass 252  
set\_resolution, YCurrent 290  
set\_resolution, YGenericSensor 567  
set\_resolution, YGyro 622  
set\_resolution, YHumidity 684  
set\_resolution, YLightSensor 751  
set\_resolution, YMagnetometer 792  
set\_resolution, YPower 1005  
set\_resolution, YPressure 1043  
set\_resolution, YPwmInput 1090  
set\_resolution, YQt 1187  
set\_resolution, YSensor 1321  
set\_resolution, YTemperature 1450  
set\_resolution, YTilt 1489  
set\_resolution, YVoc 1527  
set\_resolution, YVoltage 1565  
set\_rgbColor, YColorLed 212  
set\_rgbColorAtPowerOn, YColorLed 213  
set\_RTS, YSerialPort 1368  
set\_running, YWatchdog 1706  
set\_secondaryDNS, YNetwork 937  
set\_sensitivity, YAnButton 147  
set\_sensorType, YTemperature 1451  
set\_serialMode, YSerialPort 1371  
set\_signalBias, YGenericSensor 568  
set\_signalRange, YGenericSensor 569  
set\_sleepCountdown, YWakeUpMonitor 1626  
set\_starterTime, YMotor 885  
set\_startupSeq, YDisplay 438  
set\_state, YRelay 1282  
set\_state, YWatchdog 1707  
set\_stateAtPowerOn, YRelay 1283  
set\_stateAtPowerOn, YWatchdog 1708  
set\_timeUTC, YDataLogger 323  
set\_triggerDelay, YWatchdog 1709  
set\_triggerDuration, YWatchdog 1710  
set\_unit, YGenericSensor 570  
set\_unixTime, YRealTimeClock 1212  
set\_userData, YAccelerometer 70  
set\_userData, YAltitude 111  
set\_userData, YAnButton 148  
set\_userData, YCarbonDioxide 186  
set\_userData, YColorLed 214  
set\_userData, YCompass 253  
set\_userData, YCurrent 291  
set\_userData, YDataLogger 324  
set\_userData, YDigitalIO 395  
set\_userData, YDisplay 439  
set\_userData, YDualPower 498  
set\_userData, YFiles 524  
set\_userData, YGenericSensor 571  
set\_userData, YGyro 623

set(userData, YHubPort 647  
set(userData, YHumidity 685  
set(userData, YLed 711  
set(userData, YLightSensor 752  
set(userData, YMagnetometer 793  
set(userData, YModule 843  
set(userData, YMotor 886  
set(userData, YNetwork 938  
set(userData, YOsControl 963  
set(userData, YPower 1006  
set(userData, YPressure 1044  
set(userData, YPwmInput 1091  
set(userData, YPwmOutput 1128  
set(userData, YPwmPowerSource 1150  
set(userData, YQt 1188  
set(userData, YRealTimeClock 1213  
set(userData, YRefFrame 1248  
set(userData, YRelay 1284  
set(userData, YSensor 1322  
set(userData, YSerialPort 1372  
set(userData, YServo 1412  
set(userData, YTemperature 1452  
set(userData, YTilt 1490  
set(userData, YVoc 1528  
set(userData, YVoltage 1566  
set(userData, YVSource 1595  
set(userData, YWakeUpMonitor 1627  
set(userData, YWakeUpSchedule 1666  
set(userData, YWatchdog 1711  
set(userData, YWireless 1739  
set(userPassword, YNetwork 939  
set(userVar, YModule 844  
set\_utcOffset, YRealTimeClock 1214  
set\_valueRange, YGenericSensor 572  
set\_voltage, YVSource 1596  
set\_weekDays, YWakeUpSchedule 1667  
set\_wwwWatchdogDelay, YNetwork 940  
setAntialiasingMode, YDisplayLayer 469  
setConsoleBackground, YDisplayLayer 470  
setConsoleMargins, YDisplayLayer 471  
setConsoleWordWrap, YDisplayLayer 472  
setLayerPosition, YDisplayLayer 473  
shutdown, YOsControl 964  
Sleep, YAPI 26  
sleep, YWakeUpMonitor 1628  
sleepFor, YWakeUpMonitor 1629  
sleepUntil, YWakeUpMonitor 1630  
softAPNetwork, YWireless 1740  
Source 1568  
start3DCalibration, YRefFrame 1249  
stopSequence, YDisplay 440  
swapLayerContent, YDisplay 441

## T

Temperature 1414  
Temps 1190  
Tension 1568  
Tilt 1454  
toggle\_bitState, YDigitalIO 396

triggerFirmwareUpdate, YModule 845  
TriggerHubDiscovery, YAPI 27  
Type 1286

## U

unhide, YDisplayLayer 474  
UnregisterHub, YAPI 28  
UpdateDeviceList, YAPI 29  
updateFirmware, YModule 846  
upload, YDisplay 442  
upload, YFiles 525  
useDHCP, YNetwork 941  
useStaticIP, YNetwork 942

## V

Valeur 795  
Visual 3  
VisualBasic 3  
Voltage 1530  
voltageMove, YVSource 1597

## W

wakeUp, YWakeUpMonitor 1631  
WakeUpMonitor 1599  
WakeUpSchedule 1633  
Watchdog 1669  
Wireless 1713  
writeArray, YSerialPort 1373  
writeBin, YSerialPort 1374  
writeHex, YSerialPort 1375  
writeLine, YSerialPort 1376  
writeMODBUS, YSerialPort 1377  
writeStr, YSerialPort 1378

## Y

YAccelerometer 33-70  
YAltitude 74-111  
YAnButton 115-148  
YAPI 12-29  
YCarbonDioxide 152-186  
yCheckLogicalName 12  
YColorLed 189-214  
YCompass 218-253  
YCurrent 257-291  
YDataLogger 295-324  
YDataRun 326  
YDataSet 329-338  
YDataStream 341-353  
YDigitalIO 357-396  
yDisableExceptions 13  
YDisplay 400-442  
YDisplayLayer 445-474  
YDualPower 477-498  
yEnableExceptions 14  
YFiles 501-525  
yFindAccelerometer 33  
yFindAltitude 74

yFindAnButton 115  
yFindCarbonDioxide 152  
yFindColorLed 189  
yFindCompass 218  
yFindCurrent 257  
yFindDataLogger 295  
yFindDigitalIO 357  
yFindDisplay 400  
yFindDualPower 477  
yFindFiles 501  
yFindGenericSensor 529  
yFindGyro 577  
yFindHubPort 626  
yFindHumidity 651  
yFindLed 688  
yFindLightSensor 715  
yFindMagnetometer 756  
yFindModule 803  
yFindMotor 850  
yFindNetwork 891  
yFindOsControl 945  
yFindPower 968  
yFindPressure 1010  
yFindPwmInput 1048  
yFindPwmOutput 1095  
yFindPwmPowerSource 1131  
yFindQt 1154  
yFindRealTimeClock 1191  
yFindRefFrame 1218  
yFindRelay 1253  
yFindSensor 1288  
yFindSerialPort 1327  
yFindServo 1382  
yFindTemperature 1416  
yFindTilt 1456  
yFindVoc 1494  
yFindVoltage 1532  
yFindVSource 1569  
yFindWakeUpMonitor 1601  
yFindWakeUpSchedule 1635  
yFindWatchdog 1671  
yFindWireless 1714  
yFirstAccelerometer 34  
yFirstAltitude 75  
yFirstAnButton 116  
yFirstCarbonDioxide 153  
yFirstColorLed 190  
yFirstCompass 219  
yFirstCurrent 258  
yFirstDataLogger 296  
yFirstDigitalIO 358  
yFirstDisplay 401  
yFirstDualPower 478  
yFirstFiles 502  
yFirstGenericSensor 530  
yFirstGyro 578  
yFirstHubPort 627  
yFirstHumidity 652  
yFirstLed 689  
yFirstLightSensor 716  
yFirstMagnetometer 757  
yFirstModule 804  
yFirstMotor 851  
yFirstNetwork 892  
yFirstOsControl 946  
yFirstPower 969  
yFirstPressure 1011  
yFirstPwmInput 1049  
yFirstPwmOutput 1096  
yFirstPwmPowerSource 1132  
yFirstQt 1155  
yFirstRealTimeClock 1192  
yFirstRefFrame 1219  
yFirstRelay 1254  
yFirstSensor 1289  
yFirstSerialPort 1328  
yFirstServo 1383  
yFirstTemperature 1417  
yFirstTilt 1457  
yFirstVoc 1495  
yFirstVoltage 1533  
yFirstVSource 1570  
yFirstWakeUpMonitor 1602  
yFirstWakeUpSchedule 1636  
yFirstWatchdog 1672  
yFirstWireless 1715  
yFreeAPI 15  
YGenericSensor 529-573  
yGetAPIVersion 16  
yGetTickCount 17  
YGyro 577-623  
yHandleEvents 18  
YHubPort 626-647  
YHumidity 651-685  
yInitAPI 19  
YLed 688-711  
YLightSensor 715-752  
YMagnetometer 756-793  
YMeasure 795-799  
YModule 803-846  
YMotor 850-886  
YNetwork 891-942  
Yocto-Demo 3  
Yocto-hub 625  
YOscControl 945-964  
YPower 968-1006  
yPreregisterHub 20  
YPressure 1010-1044  
YPwmInput 1048-1091  
YPwmOutput 1095-1128  
YPwmPowerSource 1131-1150  
YQt 1154-1188  
YRealTimeClock 1191-1214  
YRefFrame 1218-1249  
yRegisterDeviceArrivalCallback 21  
yRegisterDeviceRemovalCallback 22  
yRegisterHub 23  
yRegisterHubDiscoveryCallback 24

yRegisterLogFunction 25  
YRelay 1253-1284  
YSensor 1288-1322  
YSerialPort 1327-1378  
YServo 1382-1412  
ySleep 26  
YTemperature 1416-1452  
YTilt 1456-1490  
yTriggerHubDiscovery 27  
yUnregisterHub 28  
yUpdateDeviceList 29

YVoc 1494-1528  
YVoltage 1532-1566  
YVSource 1569-1597  
YWakeUpMonitor 1601-1631  
YWakeUpSchedule 1635-1667  
YWatchdog 1671-1711  
YWireless 1714-1740

## Z

zeroAdjust, YGenericSensor 573