



Référence de l'API JavaScript

Table des matières

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

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

1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie JavaScript de Yoctopuce pour interfacier vos senseurs et contrôleurs USB.

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

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

2. Utilisation du Yocto-Demo en Javascript

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

Au moment de l'écriture de ce manuel, la librairie Javascript fonctionne avec n'importe quel browser récent... sauf Opera. Il est probable que qu'Opera finira un jour par fonctionner avec la librairie Yoctopuce¹, mais pour l'instant ce n'est pas le cas.

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

2.1. Préparation

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

- La librairie de programmation pour Javascript²
- Le programme VirtualHub³ pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

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

2.2. Contrôle de la fonction Led

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

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

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

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

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

² www.yoctopuce.com/FR/libraries.php

³ www.yoctopuce.com/FR/virtualhub.php

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.js et yocto_led.js

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

yRegisterHub

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

yFindLed

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

```
var led = yFindLed("YCTOPOC1-123456.led");
var led = yFindLed("YCTOPOC1-123456.MaFonction");
var led = yFindLed("MonModule.led");
var led = yFindLed("MonModule.MaFonction");
var led = yFindLed("MaFonction");
</pre>
```

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

isOnline

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

set_power

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

Un exemple réel

Ouvrez votre éditeur de texte préféré⁴, recopiez le code ci-dessous, sauvez-le dans le même répertoire que les fichiers de la librairie, et ouvrez-le avec votre browser favori (sauf Opera). Vous trouverez aussi ce code dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce.

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

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

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

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

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

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

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

var led;

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

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

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

2.3. Contrôle de la partie module

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

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

```

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

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

var module;

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

    module = yFindModule(serial);
    if(module.isOnline()) {
        document.getElementById('msg').value = '';
        var html = 'serial: '+module.get_serialNumber()+'<br>';
        html += 'logical name: '+module.get_logicalName()+'<br>';
        html += 'luminosity: '+module.get_luminosity()+'%<br>';
        html += 'beacon: ';
        if (module.get_beacon()==Y_BEACON_ON)
            html+="ON <a href='javascript:beacon(Y_BEACON_OFF)'>switch off</a><br>";
        else
            html+="OFF <a href='javascript:beacon(Y_BEACON_ON)'>switch on</a><br>";

        html += 'upTime: '+parseInt(module.get_upTime()/1000)+' sec<br>';
        html += 'USB current: '+module.get_usbCurrent()+' mA<br>';
        html += 'logs:<br><pre>'+module.get_lastLogs()+'</pre><br>';
        document.getElementById('data').innerHTML = html;
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout('refresh()',1000);
}

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

```

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

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

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

var module;

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

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

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

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

Énumération des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `nextModule()` de cet objet pour

trouver les modules suivants, et ce tant que la réponse n'est pas un NULL. Ci-dessous un petit exemple listant les module connectés

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

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

  function refresh()
  {
    yUpdateDeviceList();

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

2.4. Gestion des erreurs

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

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

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

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

- Sinon... l'exception va crasher votre programme, boum!

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `yDisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

3. Reference

3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

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. Référence

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 appelée 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 enregistré avec RegisterHub.

yUpdateDeviceList(errmsg)

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

yUpdateDeviceList_async(callback, context)

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

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

YAPI

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

```
function yCheckLogicalName( name)
```

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

Paramètres :

name une chaîne de caractères contenant le nom vérifier.

Retourne :

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

YAPI.DisableExceptions()

YAPI

yDisableExceptions()yDisableExceptions()

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

```
function yDisableExceptions( )
```

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

YAPI.EnableExceptions()
yEnableExceptions()

YAPI

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

```
function yEnableExceptions( )
```

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

YAPI.FreeAPI() yFreeAPI()yFreeAPI()

YAPI

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

```
function yFreeAPI( )
```

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

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

YAPI

Retourne la version de la librairie Yoctopuce utilisée.

```
function yGetAPIVersion( )
```

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

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

Retourne :

une chaîne de caractères décrivant la version de la librairie.

YAPI.GetTickCount() yGetTickCount()yGetTickCount()

YAPI

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

```
function yGetTickCount( )
```

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

Retourne :

un long entier contenant la valeur du compteur de millisecondes.

YAPI.HandleEvents() yHandleEvents()yHandleEvents()

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

```
function yHandleEvents( errmsg)
```

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

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

Paramètres :

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

Retourne :

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

YAPI.InitAPI() yInitAPI()/yInitAPI()

Initialise la librairie de programmation de Yoctopuce explicitement.

```
function yInitAPI( mode, errmsg)
```

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

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

Paramètres :

- mode** un entier spécifiant le type de détection automatique de modules à utiliser. Les valeurs possibles sont `Y_DETECT_NONE`, `Y_DETECT_USB`, `Y_DETECT_NET` et `Y_DETECT_ALL`.
- errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

Retourne :

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

YAPI.PreregisterHub() yPreregisterHub(yPreregisterHub())

YAPI

Alternative plus tolérante à RegisterHub().

```
function yPreregisterHub( url, errmsg)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterDeviceArrivalCallback()
yRegisterDeviceArrivalCallback()
yRegisterDeviceArrivalCallback()

YAPI

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

```
function yRegisterDeviceArrivalCallback( arrivalCallback)
```

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

Paramètres :

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

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

YAPI

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

```
function yRegisterDeviceRemovalCallback( removalCallback)
```

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

Paramètres :

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

YAPI.RegisterHub() yRegisterHub()yRegisterHub()

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

```
function yRegisterHub( url, errmsg)
```

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

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

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

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

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

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

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.SetTimeout() ySetTimeout()ySetTimeout()

YAPI

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

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

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

Paramètres :

- callback** la fonction à appeler lorsque le temps d'attente est écoulé. Sous Microsoft Internet Explorer, le callback doit être spécifié sous forme d'une string à évaluer.
- ms_timeout** un entier correspondant à la durée de l'attente, en millisecondes
- arguments** des arguments supplémentaires peuvent être fournis, pour être passés à la fonction de callback si nécessaire (pas supporté sous Microsoft Internet Explorer).

Retourne :

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

YAPI.Sleep() ySleep()ySleep()

YAPI

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

```
function ySleep( ms_duration, errmsg)
```

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

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

Paramètres :

ms_duration un entier correspondant à la durée de la pause, en millisecondes

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

Retourne :

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

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

YAPI

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

```
function yUnregisterHub( url)
```

Paramètres :

url une chaîne de caractères contenant "usb" ou

YAPI.UpdateDeviceList() yUpdateDeviceList(yUpdateDeviceList())

YAPI

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

```
function yUpdateDeviceList( errmsg)
```

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

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

Paramètres :

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

Retourne :

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

YAPI.UpdateDeviceList_async() yUpdateDeviceList_async() yUpdateDeviceList_async()

YAPI

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

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

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

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

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

Paramètres :

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

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

Retourne :

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

3.2. Interface de la fonction Accelerometer

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindAccelerometer(func)

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

yFirstAccelerometer()

Commence l'énumération des accéléromètres accessibles par la librairie.

Méthodes des objets YAccelerometer

accelerometer→calibrateFromPoints(rawValues, refValues)

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

accelerometer→describe()

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

accelerometer→get_advertisedValue()

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

accelerometer→get_currentRawValue()

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

accelerometer→get_currentValue()

Retourne la valeur actuelle de l'accélération.

accelerometer→get_errorMessage()

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

accelerometer→get_errorType()

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

accelerometer→get_friendlyName()

Retourne un identifiant global de l'accéléromètre au format `NOM_MODULE . NOM_FONCTION`.

accelerometer→get_functionDescriptor()

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

accelerometer→get_functionId()

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

accelerometer→get_hardwareId()

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

accelerometer→get_highestValue()

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

accelerometer→get_logFrequency()

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

accelerometer→get_logicalName()

Retourne le nom logique de l'accéléromètre.

accelerometer→get_lowestValue()

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

accelerometer→get_module()

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

accelerometer→get_module_async(callback, context)

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

accelerometer→get_recordedData(startTime, endTime)

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

accelerometer→get_reportFrequency()

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

accelerometer→get_resolution()

Retourne la résolution des valeurs mesurées.

accelerometer→get_unit()

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

accelerometer→get_userData()

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

accelerometer→get_xValue()

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

accelerometer→get_yValue()

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

accelerometer→get_zValue()

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

accelerometer→isOnline()

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

accelerometer→isOnline_async(callback, context)

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

accelerometer→load(msValidity)

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

accelerometer→loadCalibrationPoints(rawValues, refValues)

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

accelerometer→load_async(msValidity, callback, context)

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

accelerometer→nextAccelerometer()

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

accelerometer→registerTimedReportCallback(callback)

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

3. Reference

accelerometer→**registerValueCallback**(callback)

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

accelerometer→**set_highestValue**(newval)

Modifie la mémoire de valeur maximale observée.

accelerometer→**set_logFrequency**(newval)

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

accelerometer→**set_logicalName**(newval)

Modifie le nom logique de l'accéléromètre.

accelerometer→**set_lowestValue**(newval)

Modifie la mémoire de valeur minimale observée.

accelerometer→**set_reportFrequency**(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

accelerometer→**set_resolution**(newval)

Modifie la résolution des valeurs physique mesurées.

accelerometer→**set_userData**(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( func )
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YAccelerometer.FirstAccelerometer() yFirstAccelerometer(yFirstAccelerometer())

YAccelerometer

Commence l'énumération des accéléromètres accessibles par la librairie.

```
function yFirstAccelerometer( )
```

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

Retourne :

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

accelerometer→calibrateFromPoints() 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.

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

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'accéléromètre (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

accelerometer→**get_advertisedValue()****YAccelerometer****accelerometer**→**advertisedValue()****accelerometer.get_advertisedValue()**

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

```
function get_advertisedValue( )
```

Retourne :

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

accelerometer→**get_currentRawValue()**

YAccelerometer

accelerometer→**currentRawValue()**

accelerometer.get_currentRawValue()

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

```
function get_currentRawValue( )
```

Retourne :

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

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

accelerometer→**get_currentValue()****YAccelerometer****accelerometer**→**currentValue()****accelerometer.get_currentValue()**

Retourne la valeur actuelle de l'accélération.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de l'accélération

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

accelerometer→**get_errorMessage()**

YAccelerometer

accelerometer→**errorMessage()**

accelerometer.get_errorMessage()

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

```
function get_errorMessage( )
```

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

Retourne :

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

accelerometer→**get_errorType()****YAccelerometer****accelerometer**→**errorType()****accelerometer.get_errorType()**

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

```
function get_errorType( )
```

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

Retourne :

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

accelerometer→**get_friendlyName()**

YAccelerometer

accelerometer→**friendlyName()**

accelerometer.get_friendlyName()

Retourne un identifiant global de l'accéléromètre au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

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

Retourne :

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

accelerometer→**get_functionDescriptor()**
accelerometer→**functionDescriptor()**
accelerometer.get_functionDescriptor()

YAccelerometer

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

```
function get_functionDescriptor( )
```

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

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

accelerometer→**get_functionId()**

YAccelerometer

accelerometer→**functionId()**

accelerometer.get_functionId()

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

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

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

accelerometer→**get_hardwareId()****YAccelerometer****accelerometer**→**hardwareId()****accelerometer.get_hardwareId()**

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

```
function get_hardwareId( )
```

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

Retourne :

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

accelerometer→**get_highestValue()**
accelerometer→**highestValue()**
accelerometer.get_highestValue()

YAccelerometer

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

```
function get_highestValue( )
```

Retourne :

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

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

accelerometer→**get_logFrequency()****YAccelerometer****accelerometer**→**logFrequency()****accelerometer.get_logFrequency()**

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

```
function get_logFrequency( )
```

Retourne :

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

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

accelerometer→**get_logicalName()**

YAccelerometer

accelerometer→**logicalName()**

accelerometer.get_logicalName()

Retourne le nom logique de l'accéléromètre.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'accéléromètre. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

accelerometer→get_lowestValue()**YAccelerometer****accelerometer→lowestValue()****accelerometer.get_lowestValue()**

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

```
function get_lowestValue( )
```

Retourne :

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

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

accelerometer→**get_module()**

YAccelerometer

accelerometer→**module()****accelerometer.get_module()**

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

```
function get_module( )
```

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

Retourne :

une instance de `YModule`

accelerometer→**get_module_async()****YAccelerometer****accelerometer**→**module_async()****accelerometer.get_module_async()**

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

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

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

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

Paramètres :

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

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

Retourne :

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

accelerometer→**get_recordedData()**

YAccelerometer

accelerometer→**recordedData()**

accelerometer.get_recordedData()

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

```
function get_recordedData( startTime, endTime)
```

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

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

Paramètres :

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

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

Retourne :

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

accelerometer→**get_reportFrequency()****YAccelerometer****accelerometer**→**reportFrequency()****accelerometer.get_reportFrequency()**

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

```
function get_reportFrequency( )
```

Retourne :

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

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

accelerometer→**get_resolution()**

YAccelerometer

accelerometer→**resolution()**

accelerometer.get_resolution()

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

accelerometer→**get_unit()****YAccelerometer****accelerometer**→**unit()****accelerometer.get_unit()**

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

```
function get_unit( )
```

Retourne :

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

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

accelerometer→get_userData()
accelerometer→userData()
accelerometer.get_userData()

YAccelerometer

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

```
function get_userData( )
```

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

Retourne :

l'objet stocké précédemment par l'appelant.

accelerometer→**get_xValue()****YAccelerometer****accelerometer**→**xValue()****accelerometer.get_xValue()**

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

```
function get_xValue( )
```

Retourne :

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

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

accelerometer→**get_yValue()**

YAccelerometer

accelerometer→**yValue()****accelerometer.get_yValue()**

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

```
function get_yValue( )
```

Retourne :

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

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

accelerometer→**get_zValue()****YAccelerometer****accelerometer**→**zValue()****accelerometer.get_zValue()**

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

```
function get_zValue( )
```

Retourne :

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

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

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

YAccelerometer

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

fonction **isOnline()** ()

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

Retourne :

`true` si l'accéléromètre est joignable, `false` sinon

accelerometer→**isOnline_async()**
accelerometer.isOnline_async()

YAccelerometer

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

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

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

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

Paramètres :

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

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

Retourne :

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

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

```
function load( msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→load_async() accelerometer.load_async()

YAccelerometer

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

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

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

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

Paramètres :

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

Retourne :

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

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

YAccelerometer

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

```
function nextAccelerometer( )
```

Retourne :

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

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

YAccelerometer

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

```
function registerTimedReportCallback( callback)
```

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

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

Paramètres :

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

`accelerometer`→`set_highestValue()`

YAccelerometer

`accelerometer`→`setHighestValue()`

`accelerometer.set_highestValue()`

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→set_logFrequency()**YAccelerometer****accelerometer→setLogFrequency()****accelerometer.set_logFrequency()**

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

```
function set_logFrequency( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**set_logicalName()**

YAccelerometer

accelerometer→**setLogicalName()**

accelerometer.set_logicalName()

Modifie le nom logique de l'accéléromètre.

```
function set_logicalName( newval)
```

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

Paramètres :

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

Retourne :

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

accelerometer→**set_lowestValue()**
accelerometer→**setLowestValue()**
accelerometer.set_lowestValue()

YAccelerometer

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**set_reportFrequency()**

YAccelerometer

accelerometer→**setReportFrequency()**

accelerometer.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**set_resolution()**
accelerometer→**setResolution()**
accelerometer.set_resolution()

YAccelerometer

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**set_userdata()**

YAccelerometer

accelerometer→**setUserData()**

accelerometer.set_userdata()

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

```
function set_userdata( data)
```

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

Paramètres :

data objet quelconque à mémoriser

accelerometer→**wait_async()**
accelerometer.wait_async()

YAccelerometer

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

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

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

Paramètres :

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

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

Retourne :

rien du tout :

3.3. Interface de la fonction AnButton

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindAnButton(func)

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

yFirstAnButton()

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

Méthodes des objets YAnButton

anbutton→describe()

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

anbutton→get_advertisedValue()

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

anbutton→get_analogCalibration()

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

anbutton→get_calibratedValue()

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

anbutton→get_calibrationMax()

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

anbutton→get_calibrationMin()

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

anbutton→get_errorMessage()

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

anbutton→get_errorType()

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

anbutton→get_friendlyName()

Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE . NOM_FONCTION`.

anbutton→get_functionDescriptor()

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

anbutton→**get_functionId()**

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

anbutton→**get_hardwareId()**

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

anbutton→**get_isPressed()**

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

anbutton→**get_lastTimePressed()**

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

anbutton→**get_lastTimeReleased()**

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

anbutton→**get_logicalName()**

Retourne le nom logique de l'entrée analogique.

anbutton→**get_module()**

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

anbutton→**get_module_async(callback, context)**

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

anbutton→**get_pulseCounter()**

Retourne la valeur du compteur d'impulsions.

anbutton→**get_pulseTimer()**

Retourne le timer du compteur d'impulsions (ms)

anbutton→**get_rawValue()**

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).

anbutton→**get_sensitivity()**

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

anbutton→**get_userData()**

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

anbutton→**isOnline()**

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

anbutton→**isOnline_async(callback, context)**

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

anbutton→**load(msValidity)**

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

anbutton→**load_async(msValidity, callback, context)**

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

anbutton→**nextAnButton()**

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

anbutton→**registerValueCallback(callback)**

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

anbutton→**resetCounter()**

réinitialise le compteur d'impulsions et son timer

anbutton→**set_analogCalibration(newval)**

Enclenche ou déclenche le procédure de calibration.

anbutton→**set_calibrationMax(newval)**

3. Reference

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

anbutton→**set_calibrationMin(newval)**

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

anbutton→**set_logicalName(newval)**

Modifie le nom logique de l'entrée analogique.

anbutton→**set_sensitivity(newval)**

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

anbutton→**set_userData(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( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YAnButton.FirstAnButton() yFirstAnButton()yFirstAnButton()

YAnButton

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

```
function yFirstAnButton( )
```

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

Retourne :

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

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

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

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'entrée analogique (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

anbutton→**get_advertisedValue()**

YAnButton

anbutton→**advertisedValue()**

anbutton.get_advertisedValue()

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

```
function get_advertisedValue( )
```

Retourne :

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

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

anbutton→**get_analogCalibration()****YAnButton****anbutton**→**analogCalibration()****anbutton.get_analogCalibration()**

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

```
function get_analogCalibration( )
```

Retourne :

soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

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

anbutton→**get_calibratedValue()**

YAnButton

anbutton→**calibratedValue()**

anbutton.get_calibratedValue()

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

```
function get_calibratedValue( )
```

Retourne :

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

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

anbutton→**get_calibrationMax()****YAnButton****anbutton**→**calibrationMax()****anbutton.get_calibrationMax()**

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

```
function get_calibrationMax( )
```

Retourne :

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

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

anbutton→**get_calibrationMin()**
anbutton→**calibrationMin()**
anbutton.get_calibrationMin()

YAnButton

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

```
function get_calibrationMin( )
```

Retourne :

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

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

anbutton→**get_errorMessage()****YAnButton****anbutton**→**errorMessage()****anbutton.errorMessage()**

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

```
function get_errorMessage( )
```

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

Retourne :

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

anbutton→**get_errorType()**

YAnButton

anbutton→**errorType()****anbutton.get_errorType()**

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

```
function get_errorType( )
```

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

Retourne :

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

anbutton→get_friendlyName()**YAnButton****anbutton→friendlyName()****anbutton.get_friendlyName()**

Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

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

Retourne :

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

anbutton→**get_functionDescriptor()**
anbutton→**functionDescriptor()**
anbutton.get_functionDescriptor()

YAnButton

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

```
function get_functionDescriptor( )
```

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

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

anbutton→**get_functionId()****YAnButton****anbutton**→**functionId()****anbutton.get_functionId()**

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

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

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

anbutton→**get_hardwareId()**

YAnButton

anbutton→**hardwareId()****anbutton.get_hardwareId()**

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

```
function get_hardwareId( )
```

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

Retourne :

une chaîne de caractères identifiant l'entrée analogique (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

anbutton→**get_isPressed()****YAnButton****anbutton**→**isPressed()****anbutton.get_isPressed()**

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

```
function get_isPressed( )
```

Retourne :

soit `Y_ISPRESSED_FALSE`, soit `Y_ISPRESSED_TRUE`, selon vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon

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

anbutton→**get_lastTimePressed()**
anbutton→**lastTimePressed()**
anbutton.get_lastTimePressed()

YAnButton

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

```
function get_lastTimePressed( )
```

Retourne :

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

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

anbutton→**get_lastTimeReleased()****YAnButton****anbutton**→**lastTimeReleased()****anbutton.get_lastTimeReleased()**

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

```
function get_lastTimeReleased( )
```

Retourne :

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

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

anbutton→**get_logicalName()**

YAnButton

anbutton→**logicalName()****anbutton.get_logicalName()**

Retourne le nom logique de l'entrée analogique.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'entrée analogique. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

anbutton→**get_module()****YAnButton****anbutton**→**module()****anbutton.get_module()**

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

```
function get_module( )
```

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

Retourne :

une instance de `YModule`

anbutton→**get_module_async()**
anbutton→**module_async()**
anbutton.get_module_async()

YAnButton

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

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

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

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

Paramètres :

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

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

Retourne :

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

anbutton→**get_pulseCounter()**
anbutton→**pulseCounter()**
anbutton.get_pulseCounter()

YAnButton

Retourne la valeur du compteur d'impulsions.

```
function get_pulseCounter( )
```

Retourne :

un entier représentant la valeur du compteur d'impulsions

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

anbutton→**get_pulseTimer()**

YAnButton

anbutton→**pulseTimer()****anbutton.get_pulseTimer()**

Retourne le timer du compteur d'impulsions (ms)

```
function get_pulseTimer( )
```

Retourne :

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

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

anbutton→**get_rawValue()****YAnButton****anbutton**→**rawValue()****anbutton.get_rawValue()**

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).

```
function get_rawValue( )
```

Retourne :

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

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

anbutton→**get_sensitivity()**

YAnButton

anbutton→**sensitivity()****anbutton.get_sensitivity()**

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

```
function get_sensitivity( )
```

Retourne :

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

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

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

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

```
function get_userData( )
```

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

Retourne :

l'objet stocké précédemment par l'appelant.

anbutton→**isOnline()****anbutton.isOnline()**

YAnButton

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

```
function isOnline( )
```

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

Retourne :

`true` si l'entrée analogique est joignable, `false` sinon

anbutton→**isOnline_async()**
anbutton.isOnline_async()

YAnButton

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

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

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

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

Paramètres :

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

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

Retourne :

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

anbutton→load()anbutton.load()

YAnButton

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

```
function load( msValidity)
```

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

Paramètres :

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

Retourne :

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

anbutton→**load_async()**(**anbutton.load_async()**)**YAnButton**

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

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

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

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

Paramètres :

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

Retourne :

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

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

YAnButton

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

```
function nextAnButton( )
```

Retourne :

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

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

YAnButton

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

```
function registerValueCallback( callback)
```

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

Paramètres :

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

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

YAnButton

réinitialise le compteur d'impulsions et son timer

```
function resetCounter( )
```

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

anbutton→**set_analogCalibration()**
anbutton→**setAnalogCalibration()**
anbutton.set_analogCalibration()

YAnButton

Enclenche ou déclenche le procédé de calibration.

```
function set_analogCalibration( newval)
```

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

Paramètres :

newval soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

Retourne :

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

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

anbutton→**set_calibrationMax()**

YAnButton

anbutton→**setCalibrationMax()**

anbutton.set_calibrationMax()

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

```
function set_calibrationMax( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

anbutton→**set_calibrationMin()****YAnButton****anbutton**→**setCalibrationMin()****anbutton.set_calibrationMin()**

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

```
function set_calibrationMin( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

anbutton→**set_logicalName()**
anbutton→**setLogicalName()**
anbutton.set_logicalName()

YAnButton

Modifie le nom logique de l'entrée analogique.

```
function set_logicalName( newval)
```

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

Paramètres :

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

Retourne :

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

anbutton→**set_sensitivity()****YAnButton****anbutton**→**setSensitivity()****anbutton.set_sensitivity()**

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

```
function set_sensitivity( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

anbutton→**set_userdata()**

YAnButton

anbutton→**setUserData()****anbutton.set_userdata()**

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

```
function set_userdata( data)
```

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

Paramètres :

data objet quelconque à mémoriser

anbutton→**wait_async()****anbutton.wait_async()****YAnButton**

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

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

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

Paramètres :

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

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

Retourne :

rien du tout :

3.4. Interface de la fonction CarbonDioxide

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindCarbonDioxide(func)

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

yFirstCarbonDioxide()

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

Méthodes des objets YCarbonDioxide

carbondioxide→calibrateFromPoints(rawValues, refValues)

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

carbondioxide→describe()

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

carbondioxide→get_advertisedValue()

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

carbondioxide→get_currentRawValue()

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

carbondioxide→get_currentValue()

Retourne la valeur actuelle du taux de CO2.

carbondioxide→get_errorMessage()

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

carbondioxide→get_errorType()

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

carbondioxide→get_friendlyName()

Retourne un identifiant global du capteur de CO2 au format `NOM_MODULE . NOM_FONCTION`.

carbondioxide→get_functionDescriptor()

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

carbondioxide→get_functionId()

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

carbondioxide→get_hardwareId()

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

carbondioxide→get_highestValue()

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

carbondioxide→get_logFrequency()

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

carbondioxide→get_logicalName()

Retourne le nom logique du capteur de CO2.

carbondioxide→get_lowestValue()

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

carbondioxide→get_module()

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

carbondioxide→get_module_async(callback, context)

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

carbondioxide→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.

carbondioxide→get_reportFrequency()

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

carbondioxide→get_resolution()

Retourne la résolution des valeurs mesurées.

carbondioxide→get_unit()

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

carbondioxide→get_userData()

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

carbondioxide→isOnline()

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

carbondioxide→isOnline_async(callback, context)

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

carbondioxide→load(msValidity)

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

carbondioxide→loadCalibrationPoints(rawValues, refValues)

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

carbondioxide→load_async(msValidity, callback, context)

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

carbondioxide→nextCarbonDioxide()

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

carbondioxide→registerTimedReportCallback(callback)

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

carbondioxide→registerValueCallback(callback)

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

carbondioxide→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

carbondioxide→set_logFrequency(newval)

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

3. Reference

carbondioxide→**set_logicalName(newval)**

Modifie le nom logique du capteur de CO2.

carbondioxide→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

carbondioxide→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

carbondioxide→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

carbondioxide→**set_userData(data)**

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

carbondioxide→**wait_async(callback, context)**

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

YCarbonDioxide.FindCarbonDioxide() yFindCarbonDioxide()yFindCarbonDioxide()

YCarbonDioxide

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

```
function yFindCarbonDioxide( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YCarbonDioxide.FirstCarbonDioxide() yFirstCarbonDioxide()yFirstCarbonDioxide()

YCarbonDioxide

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

```
function yFirstCarbonDioxide( )
```

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

Retourne :

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

carbondioxide→calibrateFromPoints() carbondioxide.calibrateFromPoints()

YCarbonDioxide

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

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

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→describe()carbondioxide.describe()

YCarbonDioxide

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

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de CO2 (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

carbondioxide→**get_advertisedValue()****YCarbonDioxide****carbondioxide**→**advertisedValue()****carbondioxide.get_advertisedValue()**

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

```
function get_advertisedValue( )
```

Retourne :

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

carbondioxide→**get_currentRawValue()**

YCarbonDioxide

carbondioxide→**currentRawValue()**

carbondioxide.get_currentRawValue()

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

```
function get_currentRawValue( )
```

Retourne :

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

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

carbondioxide→**get_currentValue()**
carbondioxide→**currentValue()**
carbondioxide.get_currentValue()

YCarbonDioxide

Retourne la valeur actuelle du taux de CO2.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle du taux de CO2

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

**carbondioxide→get_errorMessage()
carbondioxide→errorMessage()
carbondioxide.get_errorMessage()**

YCarbonDioxide

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

```
function get_errorMessage( )
```

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

Retourne :

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

carbondioxide→get_errorType()
carbondioxide→errorType()
carbondioxide.get_errorType()

YCarbonDioxide

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

```
function get_errorType( )
```

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

Retourne :

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

carbondioxide→**get_friendlyName()**
carbondioxide→**friendlyName()**
carbondioxide.get_friendlyName()

YCarbonDioxide

Retourne un identifiant global du capteur de CO2 au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

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

Retourne :

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

carbondioxide→**get_functionDescriptor()**
carbondioxide→**functionDescriptor()**
carbondioxide.get_functionDescriptor()

YCarbonDioxide

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

```
function get_functionDescriptor( )
```

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

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

carbondioxide→get_functionId()
carbondioxide→functionId()
carbondioxide.get_functionId()

YCarbonDioxide

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

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

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

carbondioxide→**get_hardwareId()**
carbondioxide→**hardwareId()**
carbondioxide.get_hardwareId()

YCarbonDioxide

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

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de CO2 (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de CO2 (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

`carbondioxide→get_highestValue()`
`carbondioxide→highestValue()`
`carbondioxide.get_highestValue()`

YCarbonDioxide

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

```
function get_highestValue( )
```

Retourne :

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

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

carbondioxide→**get_logFrequency()****YCarbonDioxide****carbondioxide**→**logFrequency()****carbondioxide.get_logFrequency()**

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

```
function get_logFrequency( )
```

Retourne :

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

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

carbondioxide→get_logicalName()
carbondioxide→logicalName()
carbondioxide.get_logicalName()

YCarbonDioxide

Retourne le nom logique du capteur de CO2.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de CO2. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

carbondioxide→**get_lowestValue()****YCarbonDioxide****carbondioxide**→**lowestValue()****carbondioxide.get_lowestValue()**

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

```
function get_lowestValue( )
```

Retourne :

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

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

carbondioxide→get_module()
carbondioxide→module()
carbondioxide.get_module()

YCarbonDioxide

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

```
function get_module( )
```

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

Retourne :

une instance de YModule

carbondioxide→**get_module_async()**
carbondioxide→**module_async()**
carbondioxide.get_module_async()

YCarbonDioxide

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

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

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

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

Paramètres :

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

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

Retourne :

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

carbondioxide→**get_recordedData()**
carbondioxide→**recordedData()**
carbondioxide.get_recordedData()

YCarbonDioxide

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

```
function get_recordedData( startTime, endTime)
```

Veillez 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.

carbondioxide→**get_reportFrequency()****YCarbonDioxide****carbondioxide**→**reportFrequency()****carbondioxide.get_reportFrequency()**

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

```
function get_reportFrequency( )
```

Retourne :

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

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

carbondioxide→**get_resolution()**
carbondioxide→**resolution()**
carbondioxide.get_resolution()

YCarbonDioxide

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

carbondioxide→**get_unit()****YCarbonDioxide****carbondioxide**→**unit()****carbondioxide.get_unit()**

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

```
function get_unit( )
```

Retourne :

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

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

carbondioxide→get_userData()
carbondioxide→userData()
carbondioxide.get_userData()

YCarbonDioxide

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

```
function get_userData( )
```

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

Retourne :

l'objet stocké précédemment par l'appelant.

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

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

```
function isOnline( )
```

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

Retourne :

`true` si le capteur de CO2 est joignable, `false` sinon

**carbondioxide→isOnline_async()
carbondioxide.isOnline_async()**

YCarbonDioxide

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

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

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

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

Paramètres :

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

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

Retourne :

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

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

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

```
function load( msValidity)
```

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

Paramètres :

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

Retourne :

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

**carbondioxide→loadCalibrationPoints()
carbondioxide.loadCalibrationPoints()**

YCarbonDioxide

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

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

Paramètres :

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

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

Retourne :

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

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

**carbondioxide→load_async()
carbondioxide.load_async()****YCarbonDioxide**

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

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

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

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

Paramètres :

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

Retourne :

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

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

YCarbonDioxide

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

```
function nextCarbonDioxide( )
```

Retourne :

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

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

YCarbonDioxide

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

```
function registerTimedReportCallback( callback )
```

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

Paramètres :

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

**carbondioxide→registerValueCallback()
carbondioxide.registerValueCallback()**

YCarbonDioxide

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

```
function registerValueCallback( callback)
```

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

Paramètres :

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

carbondioxide→**set_highestValue()**
carbondioxide→**setHighestValue()**
carbondioxide.set_highestValue()

YCarbonDioxide

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→set_logFrequency()
carbondioxide→setLogFrequency()
carbondioxide.set_logFrequency()

YCarbonDioxide

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

```
function set_logFrequency( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→**set_logicalName()****YCarbonDioxide****carbondioxide**→**setLogicalName()****carbondioxide.set_logicalName()**

Modifie le nom logique du capteur de CO2.

```
function set_logicalName( newval)
```

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

Paramètres :

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

Retourne :

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

carbondioxide→**set_lowestValue()**
carbondioxide→**setLowestValue()**
carbondioxide.set_lowestValue()

YCarbonDioxide

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

`carbondioxide`→`set_reportFrequency()`
`carbondioxide`→`setReportFrequency()`
`carbondioxide.set_reportFrequency()`

YCarbonDioxide

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→**set_resolution()**
carbondioxide→**setResolution()**
carbondioxide.set_resolution()

YCarbonDioxide

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→**set_userData()**
carbondioxide→**setUserData()**
carbondioxide.set_userData()

YCarbonDioxide

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

```
function set_userData( data)
```

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

Paramètres :

data objet quelconque à mémoriser

carbondioxide→wait_async() carbondioxide.wait_async()

YCarbonDioxide

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

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

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

Paramètres :

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

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

Retourne :

rien du tout :

3.5. Interface de la fonction ColorLed

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindColorLed(func)

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

yFirstColorLed()

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

Méthodes des objets YColorLed

colorled→describe()

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

colorled→get_advertisedValue()

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

colorled→get_errorMessage()

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

colorled→get_errorType()

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

colorled→get_friendlyName()

Retourne un identifiant global de la led RGB au format `NOM_MODULE . NOM_FONCTION`.

colorled→get_functionDescriptor()

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

colorled→get_functionId()

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

colorled→get_hardwareId()

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

colorled→get_hslColor()

Retourne la couleur HSL courante de la led.

colorled→get_logicalName()

Retourne le nom logique de la led RGB.

3. Reference

colorled→**get_module()**

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

colorled→**get_module_async(callback, context)**

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

colorled→**get_rgbColor()**

Retourne la couleur RGB courante de la led.

colorled→**get_rgbColorAtPowerOn()**

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

colorled→**get_userData()**

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

colorled→**hslMove(hsl_target, ms_duration)**

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

colorled→**isOnline()**

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

colorled→**isOnline_async(callback, context)**

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

colorled→**load(msValidity)**

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

colorled→**load_async(msValidity, callback, context)**

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

colorled→**nextColorLed()**

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

colorled→**registerValueCallback(callback)**

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

colorled→**rgbMove(rgb_target, ms_duration)**

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

colorled→**set_hslColor(newval)**

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

colorled→**set_logicalName(newval)**

Modifie le nom logique de la led RGB.

colorled→**set_rgbColor(newval)**

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

colorled→**set_rgbColorAtPowerOn(newval)**

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

colorled→**set_userData(data)**

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

colorled→**wait_async(callback, context)**

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

YColorLed.FindColorLed() yFindColorLed()yFindColorLed()

YColorLed

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

```
function yFindColorLed( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YColorLed.FirstColorLed() yFirstColorLed()yFirstColorLed()

YColorLed

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

```
function yFirstColorLed( )
```

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

Retourne :

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

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

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

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant la led RGB (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

colorled→**get_advertisedValue()**

YColorLed

colorled→**advertisedValue()**

colorled.get_advertisedValue()

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

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de la led RGB (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

colorled→**get_errorMessage()**
colorled→**errorMessage()**
colorled.get_errorMessage()

YColorLed

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

```
function get_errorMessage( )
```

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

Retourne :

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

colorled→**get_errorType()**

YColorLed

colorled→**errorType()****colorled.get_errorType()**

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

```
function get_errorType( )
```

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

Retourne :

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

colorled→**get_friendlyName()****YColorLed****colorled**→**friendlyName()****colorled.get_friendlyName()**

Retourne un identifiant global de la led RGB au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

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

Retourne :

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

colored→**get_functionDescriptor()**
colored→**functionDescriptor()**
colored.get_functionDescriptor()

YColorLed

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

```
function get_functionDescriptor( )
```

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

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

colorled→**get_functionId()****YColorLed****colorled**→**functionId()****colorled.get_functionId()**

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

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

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

`colorled`→`get_hardwareId()`

YColorLed

`colorled`→`hardwareId()``colorled.get_hardwareId()`

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

```
function get_hardwareId( )
```

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

Retourne :

une chaîne de caractères identifiant la led RGB (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

colorled→**get_hslColor()****YColorLed****colorled**→**hslColor()****colorled.get_hslColor()**

Retourne la couleur HSL courante de la led.

```
function get_hslColor( )
```

Retourne :

un entier représentant la couleur HSL courante de la led

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

colorled→**get_logicalName()**

YColorLed

colorled→**logicalName()****colorled.get_logicalName()**

Retourne le nom logique de la led RGB.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de la led RGB. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

colorled→**get_module()****YColorLed****colorled**→**module()****colorled.get_module()**

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

```
function get_module( )
```

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

Retourne :

une instance de `YModule`

colorled→**get_module_async()**
colorled→**module_async()**
colorled.get_module_async()

YColorLed

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

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

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

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

Paramètres :

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

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

Retourne :

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

colorled→**get_rgbColor()****YColorLed****colorled**→**rgbColor()****colorled.get_rgbColor()**

Retourne la couleur RGB courante de la led.

```
function get_rgbColor( )
```

Retourne :

un entier représentant la couleur RGB courante de la led

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

colorled→**get_rgbColorAtPowerOn()**
colorled→**rgbColorAtPowerOn()**
colorled.get_rgbColorAtPowerOn()

YColorLed

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

```
function get_rgbColorAtPowerOn( )
```

Retourne :

un entier représentant la couleur configurée pour être affichage à l'allumage du module

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

colorled→**get_userData()****YColorLed****colorled**→**userData()****colorled.get_userData()**

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

```
function get_userData( )
```

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

Retourne :

l'objet stocké précédemment par l'appelant.

colorled→hsIMove()colorled.hsIMove()

YColorLed

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

```
function hsIMove( hsl_target, ms_duration)
```

Paramètres :

hsl_target couleur HSL désirée à la fin de la transition

ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

```
function isOnline( )
```

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

Retourne :

`true` si la led RGB est joignable, `false` sinon

colorled→isOnline_async()colorled.isOnline_async()

YColorLed

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

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

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

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

Paramètres :

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

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

Retourne :

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

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

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

```
function load( msValidity)
```

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

Paramètres :

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

Retourne :

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

colored→load_async()colored.load_async()

YColorLed

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

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

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

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

Paramètres :

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

Retourne :

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

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

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

```
function nextColorLed( )
```

Retourne :

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

colorled→**registerValueCallback()**
colorled.registerValueCallback()

YColorLed

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

```
function registerValueCallback( callback)
```

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

Paramètres :

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

colorled→**rgbMove()****colorled.rgbMove()****YColorLed**

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

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

Paramètres :

rgb_target couleur RGB désirée à la fin de la transition

ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_hslColor()**

YColorLed

colorled→**setHslColor()****colorled.set_hslColor()**

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

```
function set_hslColor( newval)
```

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

Paramètres :

newval un entier représentant la couleur courante de la led, en utilisant une couleur HSL spécifiée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_logicalName()**
colorled→**setLogicalName()**
colorled.set_logicalName()

YColorLed

Modifie le nom logique de la led RGB.

```
function set_logicalName( newval)
```

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

Paramètres :

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

Retourne :

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

colorled→**set_rgbColor()**

YColorLed

colorled→**setRgbColor()****colorled.set_rgbColor()**

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

```
function set_rgbColor( newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_rgbColorAtPowerOn()**
colorled→**setRgbColorAtPowerOn()**
colorled.set_rgbColorAtPowerOn()

YColorLed

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

```
function set_rgbColorAtPowerOn( newval)
```

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

Paramètres :

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

Retourne :

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

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

colorled→**set_userdata()**

YColorLed

colorled→**setUserData()****colorled.set_userdata()**

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

```
function set_userdata( data)
```

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

Paramètres :

data objet quelconque à mémoriser

colorled→**wait_async()**(colorled.wait_async())**YColorLed**

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

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

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

Paramètres :

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

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

Retourne :

rien du tout :

3.6. Interface de la fonction Compass

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindCompass(func)

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

yFirstCompass()

Commence l'énumération des compas accessibles par la librairie.

Méthodes des objets YCompass

compass→calibrateFromPoints(rawValues, refValues)

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

compass→describe()

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

compass→get_advertisedValue()

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

compass→get_currentRawValue()

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

compass→get_currentValue()

Retourne la valeur actuelle du cap relatif.

compass→get_errorMessage()

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

compass→get_errorType()

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

compass→get_friendlyName()

Retourne un identifiant global du compas au format `NOM_MODULE . NOM_FONCTION`.

compass→get_functionDescriptor()

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

compass→get_functionId()

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

compass→get_hardwareId()

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

compass→**get_highestValue()**

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

compass→**get_logFrequency()**

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

compass→**get_logicalName()**

Retourne le nom logique du compas.

compass→**get_lowestValue()**

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

compass→**get_magneticHeading()**

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

compass→**get_module()**

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

compass→**get_module_async(callback, context)**

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

compass→**get_recordedData(startTime, endTime)**

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

compass→**get_reportFrequency()**

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

compass→**get_resolution()**

Retourne la résolution des valeurs mesurées.

compass→**get_unit()**

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

compass→**get_userData()**

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

compass→**isOnline()**

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

compass→**isOnline_async(callback, context)**

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

compass→**load(msValidity)**

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

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

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

compass→**load_async(msValidity, callback, context)**

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

compass→**nextCompass()**

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

compass→**registerTimedReportCallback(callback)**

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

compass→**registerValueCallback(callback)**

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

compass→**set_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

3. Référence

compass→**set_logFrequency(newval)**

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

compass→**set_logicalName(newval)**

Modifie le nom logique du compas.

compass→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

compass→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

compass→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

compass→**set_userData(data)**

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

compass→**wait_async(callback, context)**

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

YCompass.FindCompass() yFindCompass()yFindCompass()

YCompass

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

```
function yFindCompass( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

func une chaîne de caractères qui référence le compas sans ambiguïté

Retourne :

un objet de classe `YCompass` qui permet ensuite de contrôler le compas.

YCompass.FirstCompass() yFirstCompass()yFirstCompass()

YCompass

Commence l'énumération des compas accessibles par la librairie.

```
function yFirstCompass( )
```

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

Retourne :

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

compass→**calibrateFromPoints()**
compass.calibrateFromPoints()

YCompass

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

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

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le compas (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

compass→**get_advertisedValue()****YCompass****compass**→**advertisedValue()****compass.get_advertisedValue()**

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

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du compas (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

compass→**get_currentRawValue()**

YCompass

compass→**currentRawValue()**

compass.get_currentRawValue()

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

```
function get_currentRawValue( )
```

Retourne :

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

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

compass→**get_currentValue()**
compass→**currentValue()**
compass.get_currentValue()

YCompass

Retourne la valeur actuelle du cap relatif.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle du cap relatif

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

compass→**get_errorMessage()**
compass→**errorMessage()**
compass.**get_errorMessage()**

YCompass

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

```
function get_errorMessage( )
```

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

Retourne :

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

compass→**getErrorType()****YCompass****compass**→**errorType()****compass.getErrorType()**

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

```
function getErrorType( )
```

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

Retourne :

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

compass→**get_friendlyName()**

YCompass

compass→**friendlyName()**

compass.get_friendlyName()

Retourne un identifiant global du compas au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

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

Retourne :

une chaîne de caractères identifiant le compas en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

compass→**get_functionDescriptor()****YCompass****compass**→**functionDescriptor()****compass.get_functionDescriptor()**

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

```
function get_functionDescriptor( )
```

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

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

compass→**get_functionId()**

YCompass

compass→**functionId()****compass.get_functionId()**

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

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

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

compass→**get_hardwareId()****YCompass****compass**→**hardwareId()****compass.get_hardwareId()**

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

```
function get_hardwareId( )
```

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

Retourne :

une chaîne de caractères identifiant le compas (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

compass→**get_highestValue()**

YCompass

compass→**highestValue()**

compass.get_highestValue()

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

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour le cap relatif depuis le démarrage du module

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

compass→**get_logFrequency()****YCompass****compass**→**logFrequency()****compass.get_logFrequency()**

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

```
function get_logFrequency( )
```

Retourne :

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

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

compass→**get_logicalName()**

YCompass

compass→**logicalName()****compass.get_logicalName()**

Retourne le nom logique du compas.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du compas. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

compass→**get_lowestValue()****YCompass****compass**→**lowestValue()****compass.get_lowestValue()**

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

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour le cap relatif depuis le démarrage du module

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

compass→**get_magneticHeading()**

YCompass

compass→**magneticHeading()**

compass.get_magneticHeading()

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

```
function get_magneticHeading( )
```

Retourne :

une valeur numérique représentant la direction du nord magnétique, indépendamment du cap configuré

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

compass→**get_module()****YCompass****compass**→**module()****compass.get_module()**

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

```
function get_module( )
```

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

Retourne :

une instance de `YModule`

compass→**get_module_async()****YCompass****compass**→**module_async()****compass.get_module_async()**

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

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

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

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

Paramètres :

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

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

Retourne :

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

compass→**get_recordedData()****YCompass****compass**→**recordedData()****compass.get_recordedData()**

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

```
function get_recordedData( startTime, endTime)
```

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

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

Paramètres :

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

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

compass→**get_reportFrequency()**

YCompass

compass→**reportFrequency()**

compass.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

compass→**get_resolution()****YCompass****compass**→**resolution()****compass.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

compass→**get_unit()**

YCompass

compass→**unit()****compass.get_unit()**

Retourne l'unité dans laquelle le cap relatif est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le cap relatif est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

compass→**get_userData()****YCompass****compass**→**userData()****compass.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es 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.

Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.

fonction **isOnline()** ()

Si les valeurs des attributs en cache du compas sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le compas est joignable, `false` sinon

compass→**isOnline_async()**
compass.isOnline_async()

YCompass

Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du compas sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**loadCalibrationPoints()**
compass.loadCalibrationPoints()**YCompass**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

compass→**nextCompass()****compass.nextCompass()****YCompass**

Continue l'énumération des compas commencée à l'aide de `yFirstCompass()`.

```
function nextCompass( )
```

Retourne :

un pointeur sur un objet `YCompass` accessible en ligne, ou `null` lorsque l'énumération est terminée.

compass→**registerTimedReportCallback()**
compass.registerTimedReportCallback()**YCompass**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

compass→**registerValueCallback()**
compass.registerValueCallback()

YCompass

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

compass→**set_highestValue()**
compass→**setHighestValue()**
compass.set_highestValue()

YCompass

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_logFrequency()**
compass→**setLogFrequency()**
compass.set_logFrequency()

YCompass

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_logicalName()**
compass→**setLogicalName()**
compass.set_logicalName()

YCompass

Modifie le nom logique du compas.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du compas.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_lowestValue()**
compass→**setLowestValue()**
compass.set_lowestValue()

YCompass

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_reportFrequency()**
compass→**setReportFrequency()**
compass.set_reportFrequency()

YCompass

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_resolution()****YCompass****compass**→**setResolution()****compass.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_userData()**

YCompass

compass→**setUserData()****compass.set_userData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
function set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

compass→**wait_async()****compass.wait_async()****YCompass**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.7. Interface de la fonction Current

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_current.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YCurrent = yoctolib.YCurrent;</code>
php	<code>require_once('yocto_current.php');</code>
c++	<code>#include "yocto_current.h"</code>
m	<code>#import "yocto_current.h"</code>
pas	<code>uses yocto_current;</code>
vb	<code>yocto_current.vb</code>
cs	<code>yocto_current.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YCurrent;</code>
py	<code>from yocto_current import *</code>

Fonction globales

yFindCurrent(func)

Permet de retrouver un capteur de courant d'après un identifiant donné.

yFirstCurrent()

Commence l'énumération des capteurs de courant accessibles par la librairie.

Méthodes des objets YCurrent

current→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

current→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

current→get_advertisedValue()

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

current→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

current→get_currentValue()

Retourne la valeur instantanée du courant.

current→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

current→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

current→get_friendlyName()

Retourne un identifiant global du capteur de courant au format `NOM_MODULE . NOM_FONCTION`.

current→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

current→get_functionId()

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

current→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de courant au format SERIAL . FUNCTIONID.

current→get_highestValue()

Retourne la valeur maximale observée pour le courant.

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.

current→get_logicalName()

Retourne le nom logique du capteur de courant.

current→get_lowestValue()

Retourne la valeur minimale observée pour le courant.

current→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

current→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

current→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.

current→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

current→get_resolution()

Retourne la résolution des valeurs mesurées.

current→get_unit()

Retourne l'unité dans laquelle le courant est exprimée.

current→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

current→isOnline()

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

current→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

current→load(msValidity)

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

current→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

current→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

current→nextCurrent()

Continue l'énumération des capteurs de courant commencée à l'aide de yFirstCurrent().

current→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

current→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

current→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée pour le courant.

current→set_logFrequency(newval)

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

current→**set_logicalName(newval)**

Modifie le nom logique du capteur de courant.

current→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour le courant.

current→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

current→**set_resolution(newval)**

Modifie la résolution des valeurs mesurées.

current→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

current→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YCurrent.FindCurrent() yFindCurrent()yFindCurrent()

YCurrent

Permet de retrouver un capteur de courant d'après un identifiant donné.

```
function yFindCurrent( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de courant soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de courant sans ambiguïté

Retourne :

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

YCurrent.FirstCurrent()
yFirstCurrent()**yFirstCurrent()**

YCurrent

Commence l'énumération des capteurs de courant accessibles par la librairie.

```
function yFirstCurrent( )
```

Utiliser la fonction `YCurrent.nextCurrent()` pour itérer sur les autres capteurs de courant.

Retourne :

un pointeur sur un objet `YCurrent`, correspondant au premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

current→**calibrateFromPoints()**
current.calibrateFromPoints()**YCurrent**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**describe()****current.describe()****YCurrent**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de courant (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

current→**get_advertisedValue()****YCurrent****current**→**advertisedValue()****current.get_advertisedValue()**

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de courant (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

current→**get_currentRawValue()**

YCurrent

current→**currentRawValue()**

current.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

current→**get_currentValue()****YCurrent****current**→**currentValue()****current.get_currentValue()**

Retourne la valeur instantanée du courant.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur instantanée du courant

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

current→**get_errorMessage()**

YCurrent

current→**errorMessage()****current.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

current→**get_errorType()****YCurrent****current**→**errorType()****current.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

current→**get_friendlyName()**

YCurrent

current→**friendlyName()****current.get_friendlyName()**

Retourne un identifiant global du capteur de courant au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de courant si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de courant (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de courant en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

current→**get_functionDescriptor()****YCurrent****current**→**functionDescriptor()****current.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

current→**get_functionId()**

YCurrent

current→**functionId()****current.get_functionId()**

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de courant (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

current→**get_hardwareId()****YCurrent****current**→**hardwareId()****current.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de courant au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de courant (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de courant (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

current→**get_highestValue()**

YCurrent

current→**highestValue()****current.get_highestValue()**

Retourne la valeur maximale observée pour le courant.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour le courant

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

current→**get_logFrequency()****YCurrent****current**→**logFrequency()****current.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

current→**get_logicalName()**

YCurrent

current→**logicalName()****current.get_logicalName()**

Retourne le nom logique du capteur de courant.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de courant. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

current→**get_lowestValue()****YCurrent****current**→**lowestValue()****current.get_lowestValue()**

Retourne la valeur minimale observée pour le courant.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour le courant

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

current→**get_module()**

YCurrent

current→**module()****current.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

current→**get_module_async()****YCurrent****current**→**module_async()****current.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

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( startTime, endTime)
```

Veillez 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()****YCurrent****current**→**reportFrequency()****current.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

current→**get_resolution()**

YCurrent

current→**resolution()****current.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

current→**get_unit()****YCurrent****current**→**unit()****current.get_unit()**

Retourne l'unité dans laquelle le courant est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le courant est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

current→**get_userData()**

YCurrent

current→**userData()****current.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

current→**isOnline()****current.isOnline()****YCurrent**

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de courant est joignable, `false` sinon

current→isOnline_async()current.isOnline_async()

YCurrent

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

current→**load()****current.load()****YCurrent**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**loadCalibrationPoints()**
current.loadCalibrationPoints()

YCurrent

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**load_async()****current.load_async()****YCurrent**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

current→**nextCurrent()**current.nextCurrent()

YCurrent

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

```
function nextCurrent( )
```

Retourne :

un pointeur sur un objet `YCurrent` accessible en ligne, ou `null` lorsque l'énumération est terminée.

current→**registerTimedReportCallback()**
current.registerTimedReportCallback()

YCurrent

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

current→**registerValueCallback()**
current.registerValueCallback()

YCurrent

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

current→**set_highestValue()**
current→**setHighestValue()**
current.set_highestValue()

YCurrent

Modifie la mémoire de valeur maximale observée pour le courant.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour le courant

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_logFrequency()**

YCurrent

current→**setLogFrequency()**

current.set_logFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_logicalName()****YCurrent****current**→**setLogicalName()****current.set_logicalName()**

Modifie le nom logique du capteur de courant.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de courant.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_lowestValue()**

YCurrent

current→**setLowestValue()****current.set_lowestValue()**

Modifie la mémoire de valeur minimale observée pour le courant.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour le courant

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_reportFrequency()****YCurrent****current**→**setReportFrequency()****current.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_resolution()**

YCurrent

current→**setResolution()****current.set_resolution()**

Modifie la résolution des valeurs mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_userdata()****YCurrent****current**→**setUserData()****current.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

current→wait_async()current.wait_async()

YCurrent

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.8. Interface de la fonction DataLogger

Les capteurs de Yoctopuce sont équipés d'une mémoire non-volatile permettant de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La fonction DataLogger contrôle les paramètres globaux de cet enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_datalogger.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDataLogger = yoctolib.YDataLogger;
php	require_once('yocto_datalogger.php');
c++	#include "yocto_datalogger.h"
m	#import "yocto_datalogger.h"
pas	uses yocto_datalogger;
vb	yocto_datalogger.vb
cs	yocto_datalogger.cs
java	import com.yoctopuce.YoctoAPI.YDataLogger;
py	from yocto_datalogger import *

Fonction globales

yFindDataLogger(func)

Permet de retrouver un enregistreur de données d'après un identifiant donné.

yFirstDataLogger()

Commence l'énumération des enregistreurs de données accessibles par la librairie.

Méthodes des objets YDataLogger

datalogger→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

datalogger→forgetAllDataStreams()

Efface tout l'historique des mesures de l'enregistreur de données.

datalogger→get_advertisedValue()

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

datalogger→get_autoStart()

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

datalogger→get_currentRunIndex()

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

datalogger→get_dataSets()

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

datalogger→get_dataStreams(v)

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

datalogger→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

datalogger→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

datalogger→get_friendlyName()

3. Reference

Retourne un identifiant global de l'enregistreur de données au format `NOM_MODULE . NOM_FUNCTION`.

`datalogger→get_functionDescriptor()`

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

`datalogger→get_functionId()`

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

`datalogger→get_hardwareId()`

Retourne l'identifiant matériel unique de l'enregistreur de données au format `SERIAL . FUNCTIONID`.

`datalogger→get_logicalName()`

Retourne le nom logique de l'enregistreur de données.

`datalogger→get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`datalogger→get_module_async(callback, context)`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`datalogger→get_recording()`

Retourne l'état d'activation de l'enregistreur de données.

`datalogger→get_timeUTC()`

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

`datalogger→get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

`datalogger→isOnline()`

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

`datalogger→isOnline_async(callback, context)`

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

`datalogger→load(msValidity)`

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

`datalogger→load_async(msValidity, callback, context)`

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

`datalogger→nextDataLogger()`

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

`datalogger→registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

`datalogger→set_autoStart(newval)`

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

`datalogger→set_logicalName(newval)`

Modifie le nom logique de l'enregistreur de données.

`datalogger→set_recording(newval)`

Modifie l'état d'activation de l'enregistreur de données.

`datalogger→set_timeUTC(newval)`

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

`datalogger→set_userData(data)`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

`datalogger→wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDataLogger.FindDataLogger() yFindDataLogger()yFindDataLogger()

YDataLogger

Permet de retrouver un enregistreur de données d'après un identifiant donné.

```
function yFindDataLogger( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'enregistreur de données soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnline()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

Retourne :

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

YDataLogger.FirstDataLogger() yFirstDataLogger()yFirstDataLogger()

YDataLogger

Commence l'énumération des enregistreurs de données accessibles par la librairie.

```
function yFirstDataLogger( )
```

Utiliser la fonction `YDataLogger.nextDataLogger()` pour itérer sur les autres enregistreurs de données.

Retourne :

un pointeur sur un objet `YDataLogger`, correspondant au premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

datalogger→**describe()**(**datalogger.describe()**)**YDataLogger**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

```
une chaîne de caractères décrivant l'enregistreur de données (ex:  
Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)
```

datalogger→**forgetAllDataStreams()**
datalogger.forgetAllDataStreams()

YDataLogger

Efface tout l'historique des mesures de l'enregistreur de données.

```
function forgetAllDataStreams( )
```

Cette méthode remet aussi à zéro le compteur de Runs.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**get_advertisedValue()****YDataLogger****datalogger**→**advertisedValue()****datalogger.get_advertisedValue()**

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'enregistreur de données (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

datalogger→**get_autoStart()**

YDataLogger

datalogger→**autoStart()****datalogger.get_autoStart()**

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

```
function get_autoStart( )
```

Retourne :

soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

En cas d'erreur, déclenche une exception ou retourne `Y_AUTOSTART_INVALID`.

datalogger→**get_currentRunIndex()****YDataLogger****datalogger**→**currentRunIndex()****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.

```
function get_currentRunIndex( )
```

Retourne :

un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRUNINDEX_INVALID`.

datalogger→**get_dataSets()**

YDataLogger

datalogger→**dataSets()****datalogger.get_dataSets()**

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

```
function get_dataSets( )
```

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Retourne :

une liste d'objets YDataSet

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datalogger→**get_dataStreams()****YDataLogger****datalogger**→**dataStreams()****datalogger.get_dataStreams()**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

```
function get_dataStreams( v )
```

L'appelant doit passer par référence un tableau vide pour stocker les objets YDataStream, et la méthode va les remplir avec des objets décrivant les séquences de données disponibles.

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

Paramètres :

v un tableau de YDataStreams qui sera rempli avec les séquences trouvées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**get_errorMessage()**

YDataLogger

datalogger→**errorMessage()**

datalogger.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

datalogger→**get_errorType()****YDataLogger****datalogger**→**errorType()****datalogger.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

datalogger→**get_friendlyName()**

YDataLogger

datalogger→**friendlyName()**

datalogger.get_friendlyName()

Retourne un identifiant global de l'enregistreur de données au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'enregistreur de données si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'enregistreur de données (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'enregistreur de données en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

datalogger→**get_functionDescriptor()**
datalogger→**functionDescriptor()**
datalogger.get_functionDescriptor()

YDataLogger

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

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( )
```

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()****YDataLogger****datalogger**→**hardwareId()****datalogger.get_hardwareId()**

Retourne l'identifiant matériel unique de l'enregistreur de données au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'enregistreur de données (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'enregistreur de données (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

datalogger→**get_logicalName()**

YDataLogger

datalogger→**logicalName()**

datalogger.get_logicalName()

Retourne le nom logique de l'enregistreur de données.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'enregistreur de données. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

datalogger→**get_module()****YDataLogger****datalogger**→**module()****datalogger.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

datalogger→**get_module_async()****YDataLogger****datalogger**→**module_async()****datalogger.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

datalogger→**get_recording()****YDataLogger****datalogger**→**recording()****datalogger.get_recording()**

Retourne l'état d'activation de l'enregistreur de données.

```
function get_recording( )
```

Retourne :

soit `Y_RECORDING_OFF`, soit `Y_RECORDING_ON`, selon l'état d'activation de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_RECORDING_INVALID`.

datalogger→**get_timeUTC()**

YDataLogger

datalogger→**timeUTC()****datalogger.get_timeUTC()**

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

```
function get_timeUTC( )
```

Retourne :

un entier représentant le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue

En cas d'erreur, déclenche une exception ou retourne `Y_TIMEUTC_INVALID`.

datalogger→**get_userData()****YDataLogger****datalogger**→**userData()****datalogger.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es 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.

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

fonction **isOnline()** ()

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'enregistreur de données est joignable, `false` sinon

datalogger→isOnline_async()
datalogger.isOnline_async()

YDataLogger

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

datalogger→**load()**(**datalogger.load()**)

YDataLogger

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**load_async()****datalogger.load_async()****YDataLogger**

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quiel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

datalogger→**nextDataLogger()**
datalogger.nextDataLogger()

YDataLogger

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

```
function nextDataLogger( )
```

Retourne :

un pointeur sur un objet `YDataLogger` accessible en ligne, ou `null` lorsque l'énumération est terminée.

datalogger→**registerValueCallback()**
datalogger.registerValueCallback()

YDataLogger

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

datalogger→**set_autoStart()**

YDataLogger

datalogger→**setAutoStart()****datalogger.set_autoStart()**

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

```
function set_autoStart( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_logicalName()****YDataLogger****datalogger**→**setLogicalName()****datalogger.set_logicalName()**

Modifie le nom logique de l'enregistreur de données.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'enregistreur de données.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_recording()**

YDataLogger

datalogger→**setRecording()**

datalogger.set_recording()

Modifie l'état d'activation de l'enregistreur de données.

```
function set_recording( newval)
```

Paramètres :

newval soit Y_RECORDING_OFF, soit Y_RECORDING_ON, selon l'état d'activation de l'enregistreur de données

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_timeUTC()****YDataLogger****datalogger**→**setTimeUTC()****datalogger.set_timeUTC()**

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

```
function set_timeUTC( newval)
```

Paramètres :

newval un entier représentant la référence de temps UTC, afin de l'attacher aux données enregistrées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_userdata()**

YDataLogger

datalogger→**setUserData()****datalogger.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( 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

datalogger→**wait_async()****datalogger.wait_async()****YDataLogger**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.9. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_datalogger.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDataLogger = yoctolib.YDataLogger;
php	require_once('yocto_datalogger.php');
c++	#include "yocto_datalogger.h"
m	#import "yocto_datalogger.h"
pas	uses yocto_datalogger;
vb	yocto_datalogger.vb
cs	yocto_datalogger.cs
java	import com.yoctopuce.YoctoAPI.YDataLogger;
py	from yocto_datalogger import *

Méthodes des objets YDataRun

datarun→**get_averageValue(measureName, pos)**

Retourne la valeur moyenne des mesures observées au moment choisi.

datarun→**get_duration()**

Retourne la durée (en secondes) du Run.

datarun→**get_maxValue(measureName, pos)**

Retourne la valeur maximale des mesures observées au moment choisi.

datarun→**get_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

datarun→**get_minValue(measureName, pos)**

Retourne la valeur minimale des mesures observées au moment choisi.

datarun→**get_startTimeUTC()**

Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

datarun→**get_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

datarun→**get_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

datarun→**set_valueInterval(valueInterval)**

Change l'intervalle de temps représenté par chaque valeur de ce run.

datarun→**get_averageValue()**

YDataRun

datarun→**averageValue()****datarun.get_averageValue()**

Retourne la valeur moyenne des mesures observées au moment choisi.

```
function get_averageValue( measureName, pos)
```

datarun→**get_averageValue()****datarun**→**averageValue()****datarun.get_averageValue()**

Retourne la valeur moyenne des mesures observées au moment choisi.

```
js function get_averageValue( measureName, pos)  
nodejs function get_averageValue( measureName, pos)  
php function get_averageValue( $measureName, $pos)  
java double get_averageValue( String measureName, int pos)  
py def get_averageValue( measureName, pos)
```

Paramètres :

measureName le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

pos l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

Retourne :

une nombre flottant (la valeur moyenne).

En cas d'erreur, déclenche une exception ou retourne `Y_AVERAGEVALUE_INVALID`.

datarun→**get_duration()**

YDataRun

datarun→**duration()****datarun.get_duration()**

Retourne la durée (en secondes) du Run.

```
function get_duration( )
```

datarun→**get_duration()**

datarun→**duration()****datarun.get_duration()**

Retourne la durée (en secondes) du Run.

```
js function get_duration( )
```

```
nodejs function get_duration( )
```

```
php function get_duration( )
```

```
java long get_duration( )
```

```
py def get_duration( )
```

Lorsque cette méthode est appelée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le début du Run (quand le module a été mis sous tension) et la dernière mesure enregistrée.

datarun→**get_maxValue()****YDataRun****datarun**→**maxValue()****datarun.get_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

```
function get_maxValue( measureName, pos)
```

datarun→**get_maxValue()****datarun**→**maxValue()****datarun.get_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

js	function get_maxValue (measureName , pos)
nodejs	function get_maxValue (measureName , pos)
php	function get_maxValue (\$measureName , \$pos)
java	double get_maxValue (String measureName , int pos)
py	def get_maxValue (measureName , pos)

Paramètres :

measureName le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

pos l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

Retourne :

une nombre flottant (la valeur maximale).

En cas d'erreur, déclenche une exception ou retourne `Y_MAXVALUE_INVALID`.

datarun→**get_measureNames()**

YDataRun

datarun→**measureNames()**

datarun.get_measureNames()

Retourne les noms des valeurs mesurées par l'enregistreur de données.

```
function get_measureNames( )
```

datarun→**get_measureNames()**

datarun→**measureNames()****datarun.get_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

```
js function get_measureNames( )
```

```
nodejs function get_measureNames( )
```

```
php function get_measureNames( )
```

```
java ArrayList<String> get_measureNames( )
```

```
py def get_measureNames( )
```

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure.

Retourne :

une liste de chaîne de caractères (les noms des mesures)

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datarun→**get_minValue()****YDataRun****datarun**→**minValue()****datarun.get_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

```
function get_minValue( measureName, pos)
```

datarun→**get_minValue()****datarun**→**minValue()****datarun.get_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

```
js function get_minValue( measureName, pos)  
nodejs function get_minValue( measureName, pos)  
php function get_minValue( $measureName, $pos)  
java double get_minValue( String measureName, int pos)  
py def get_minValue( measureName, pos)
```

Paramètres :

measureName le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

pos l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

Retourne :

une nombre flottant (la valeur minimale).

En cas d'erreur, déclenche une exception ou retourne `Y_MINVALUE_INVALID`.

datarun→get_startTimeUTC()

YDataRun

datarun→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).

Si l'heure UTC n'a jamais été configurée dans l'enregistreur de données durant le run, et si il ne s'agit pas du run courant, cette méthode retourne 0.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début du Run.

datarun→**get_valueCount()****YDataRun****datarun**→**valueCount()****datarun.get_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

```
function get_valueCount( )
```

datarun→**get_valueCount()****datarun**→**valueCount()****datarun.get_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

```
js function get_valueCount( )
```

```
nodejs function get_valueCount( )
```

```
php function get_valueCount( )
```

```
java int get_valueCount( )
```

```
py def get_valueCount( )
```

Lorsque cette méthode est appelée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

Retourne :

un entier positif correspondant à la durée du Run divisée par l'intervalle entre les valeurs.

datarun→**get_valueInterval()**

YDataRun

datarun→**valueInterval()****datarun.get_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

```
function get_valueInterval( )
```

datarun→**get_valueInterval()**

datarun→**valueInterval()****datarun.get_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

```
js function get_valueInterval( )
```

```
nodejs function get_valueInterval( )
```

```
php function get_valueInterval( )
```

```
java int get_valueInterval( )
```

```
py def get_valueInterval( )
```

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

Retourne :

un entier positif correspondant au nombre de secondes couvertes par chaque valeur représentée dans le Run.

datarun→**set_valueInterval()**

YDataRun

datarun→**setValueInterval()****datarun.set_valueInterval()**

Change l'intervalle de temps représenté par chaque valeur de ce run.

```
function set_valueInterval( valueInterval)
```

datarun→**set_valueInterval()****datarun**→**setValueInterval()****datarun.set_valueInterval()**

Change l'intervalle de temps représenté par chaque valeur de ce run.

```
js function set_valueInterval( valueInterval)
```

```
nodejs function set_valueInterval( valueInterval)
```

```
php function set_valueInterval( $valueInterval)
```

```
java void set_valueInterval( int valueInterval)
```

```
py def set_valueInterval( valueInterval)
```

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

Paramètres :

valueInterval un nombre entier de secondes.

Retourne :

nothing

3.10. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-même sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

Cette classe ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

Méthodes des objets YDataSet

dataset→`get_endTimeUTC()`

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

dataset→`get_functionId()`

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

dataset→`get_hardwareId()`

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

dataset→`get_measures()`

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

dataset→`get_preview()`

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

dataset→`get_progress()`

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

dataset→`get_startTimeUTC()`

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

dataset→`get_summary()`

Retourne un objet YMeasure résumant tout le YDataSet.

dataset→`get_unit()`

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

dataset→loadMore()

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

dataset→loadMore_async(callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

dataset→**get_endTimeUTC()**

YDataSet

dataset→**endTimeUTC()****dataset.get_endTimeUTC()**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

```
function get_endTimeUTC( )
```

Lorsque l'objet YDataSet est créé, l'heure de fin est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de fin est mise à jour à la dernière mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la dernière mesure.

dataset→**get_functionId()****YDataSet****dataset**→**functionId()****dataset.get_functionId()**

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

```
function get_functionId( )
```

Par exemple `temperature1`.

Retourne :

une chaîne de caractères identifiant la fonction (ex: `temperature1`)

dataset→**get_hardwareId()**

YDataSet

dataset→**hardwareId()****dataset.get_hardwareId()**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple THRMCP11-123456.temperature1).

Retourne :

une chaîne de caractères identifiant la fonction (ex: THRMCP11-123456.temperature1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

dataset→**get_measures()****YDataSet****dataset**→**measures()****dataset.get_measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

```
function get_measures( )
```

Chaque élément contient: - le moment où la mesure a débuté - le moment où la mesure s'est terminée - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Avant d'appeler cette méthode, vous devez appeler `loadMore()` pour charger des données depuis l'enregistreur sur le module. L'appel doit être répété plusieurs fois pour charger toutes les données, mais vous pouvez commencer à utiliser les données disponibles avant qu'elles n'aient été toutes chargées

Les mesures les plus anciennes sont toujours chargées les premières, et les plus récentes en dernier. De ce fait, les timestamps dans la table des mesures sont normalement par ordre chronologique. La seule exception est dans le cas où il y a eu un ajustement de l'horloge UTC de l'enregistreur de données pendant l'enregistrement.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant une mesure effectuée à un moment précis.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

dataset→**get_preview()**

YDataSet

dataset→**preview()****dataset.get_preview()**

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

function **get_preview**()

Chaque élément contient: - le début d'un intervalle de temps - la fin d'un intervalle de temps - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Le résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

dataset→**get_progress()****YDataSet****dataset**→**progress()****dataset.get_progress()**

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

```
function get_progress( )
```

A l'instanciation de l'objet par la fonction `get_dataSet()`, l'avancement est nul. Au fur et à mesure des appels à `loadMore()`, l'avancement progresse pour atteindre la valeur 100 lorsque toutes les mesures ont été chargées.

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées.

dataset→**get_startTimeUTC()**

YDataSet

dataset→**startTimeUTC()****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).

```
function get_startTimeUTC() ( )
```

Lorsque l'objet YDataSet est créé, l'heure de départ est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de départ est mise à jour à la première mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la première mesure enregistrée.

dataset→**get_summary()****YDataSet****dataset**→**summary()****dataset.get_summary()**

Retourne un objet YMeasure résumant tout le YDataSet.

```
function get_summary( )
```

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

Retourne :

un objet YMeasure

dataset→**get_unit()**

YDataSet

dataset→**unit()****dataset.get_unit()**

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant une unité physique.

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

dataset→**loadMore()****dataset.loadMore()****YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

```
function loadMore( )
```

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dataset → **loadMore_async()** **dataset.loadMore_async()**

YDataSet

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
function loadMore_async( callback, context)
```

Paramètres :

callback fonction fournie par l'utilisateur, qui sera appelée lorsque la suite du chargement aura été effectué. La fonction callback doit prendre trois arguments: - la variable de contexte à disposition de l'utilisateur - l'objet YDataSet dont la méthode loadMore_async a été appelée - le résultat de l'appel: soit l'état d'avancement du chargement (0...100), ou un code d'erreur négatif en cas de problème.

context variable de contexte à disposition de l'utilisateur

Retourne :

rien.

3.11. Séquence de données enregistrées brute

Les objets YDataStream correspondent aux séquences de mesures enregistrées brutes, directement telles qu'obtenues par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Dans la plupart des cas, il n'est pas nécessaire d'utiliser les objets DataStream, car les objets YDataSet (retournés par la méthode `get_recordedData()` des senseurs et la méthode `get_dataSets()` du DataLogger) fournissent une interface plus pratique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YAPI = yoctolib.YAPI;</code> <code>var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

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→`get_rowCount()`

Retourne le nombre d'enregistrement contenus dans la séquence.

datastream→`get_runIndex()`

Retourne le numéro de Run de la séquence de données.

datastream→`get_startTime()`

3. Reference

Retourne le temps de départ relatif de la séquence (en secondes).

datastream→**get_startTimeUTC()**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

datastream→get_averageValue()**YDataStream****datastream→averageValue()****datastream.get_averageValue()**

Retourne la moyenne des valeurs observées durant cette séquence.

```
function get_averageValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y_DATA_INVALID.

Retourne :

un nombre décimal correspondant à la moyenne des valeurs, ou Y_DATA_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→**get_columnCount()**

YDataStream

datastream→**columnCount()**

datastream.get_columnCount()

Retourne le nombre de colonnes de données contenus dans la séquence.

```
function get_columnCount( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :

un entier positif correspondant au nombre de colonnes.

En cas d'erreur, déclenche une exception ou retourne zéro.

datastream→**get_columnNames()****YDataStream****datastream**→**columnNames()****datastream.get_columnNames()**

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

```
function get_columnNames( )
```

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure. Pour les séquences enregistrées à faible fréquence, l'enregistreur de donnée stocke la valeur min, moyenne et max observée durant chaque intervalle de temps dans des colonnes avec les suffixes `_min`, `_avg` et `_max` respectivement.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :

une liste de chaîne de caractères.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datastream→**get_data()**

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( row, col)
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Paramètres :

row index de l'enregistrement (ligne)

col index de la mesure (colonne)

Retourne :

un nombre décimal

En cas d'erreur, déclenche une exception ou retourne `Y_DATA_INVALID`.

datastream→**get_dataRows()****YDataStream****datastream**→**dataRows()****datastream.get_dataRows()**

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

```
function get_dataRows( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `getColumnNames()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :

une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datastream→**get_dataSamplesIntervalMs()**

YDataStream

datastream→**dataSamplesIntervalMs()**

datastream.get_dataSamplesIntervalMs()

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

```
function get_dataSamplesIntervalMs( )
```

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la fréquence d'enregistrement peut être changée pour chaque fonction.

Retourne :

un entier positif correspondant au nombre de millisecondes entre deux mesures consécutives.

datastream→**get_duration()****YDataStream****datastream**→**duration()****datastream.get_duration()**

Retourne la durée approximative de cette séquence, en secondes.

```
function get_duration( )
```

Retourne :

le nombre de secondes couvertes par cette séquence.

En cas d'erreur, déclenche une exception ou retourne Y_DURATION_INVALID.

datastream→**get_maxValue()**

YDataStream

datastream→**maxValue()****datastream.get_maxValue()**

Retourne la plus grande valeur observée durant cette séquence.

```
function get_maxValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y_DATA_INVALID.

Retourne :

un nombre décimal correspondant à la plus grande valeur, ou Y_DATA_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→**get_minValue()****YDataStream****datastream**→**minValue()****datastream.get_minValue()**

Retourne la plus petite valeur observée durant cette séquence.

```
function get_minValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y_DATA_INVALID.

Retourne :

un nombre décimal correspondant à la plus petite valeur, ou Y_DATA_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→**get_rowCount()**

YDataStream

datastream→**rowCount()****datastream.get_rowCount()**

Retourne le nombre d'enregistrement contenus dans la séquence.

```
function get_rowCount( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :

un entier positif correspondant au nombre d'enregistrements.

En cas d'erreur, déclenche une exception ou retourne zéro.

datastream→**get_runIndex()****YDataStream****datastream**→**runIndex()****datastream.get_runIndex()**

Retourne le numéro de Run de la séquence de données.

```
function get_runIndex( )
```

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

Retourne :

un entier positif correspondant au numéro du Run

datastream→**get_startTime()**

YDataStream

datastream→**startTime()****datastream.get_startTime()**

Retourne le temps de départ relatif de la séquence (en secondes).

```
function get_startTime( )
```

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 `get_startTimeUTC()`.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le début du Run et le début de la séquence enregistrée.

datastream→**get_startTimeUTC()****YDataStream****datastream**→**startTimeUTC()****datastream.get_startTimeUTC()**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

```
function get_startTimeUTC( )
```

Si l'heure UTC n'était pas configurée dans l'enregistreur de données au début de la séquence, cette méthode retourne 0.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début de la séquence enregistrée.

3.12. Interface de la fonction DigitalIO

La librairie de programmation Yoctopuce permet simplement de changer l'état de chaque bit du port d'entrée sortie. Il est possible de changer tous les bits du port à la fois, ou de les changer indépendamment. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Le comportement électrique de chaque entrée/sortie peut être modifié (open drain et polarité inverse).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_digitalio.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDigitalIO = yoctolib.YDigitalIO;
php	require_once('yocto_digitalio.php');
cpp	#include "yocto_digitalio.h"
m	#import "yocto_digitalio.h"
pas	uses yocto_digitalio;
vb	yocto_digitalio.vb
cs	yocto_digitalio.cs
java	import com.yoctopuce.YoctoAPI.YDigitalIO;
py	from yocto_digitalio import *

Fonction globales

yFindDigitalIO(func)

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

yFirstDigitalIO()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

Méthodes des objets YDigitalIO

digitalio→delayedPulse(bitno, ms_delay, ms_duration)

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

digitalio→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format TYPE (NAME) =SERIAL . FUNCTIONID.

digitalio→get_advertisedValue()

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

digitalio→get_bitDirection(bitno)

Retourne la direction d'un seul bit du port d'E/S.

digitalio→get_bitOpenDrain(bitno)

Retourne la direction d'un seul bit du port d'E/S.

digitalio→get_bitPolarity(bitno)

Retourne la polarité d'un seul bit du port d'E/S.

digitalio→get_bitState(bitno)

Retourne l'état d'un seul bit du port d'E/S.

digitalio→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

digitalio→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

digitalio→get_friendlyName()

Retourne un identifiant global du port d'E/S digital au format NOM_MODULE . NOM_FONCTION.

digitalio→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

digitalio→get_functionId()

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

digitalio→get_hardwareId()

Retourne l'identifiant matériel unique du port d'E/S digital au format `SERIAL . FUNCTIONID`.

digitalio→get_logicalName()

Retourne le nom logique du port d'E/S digital.

digitalio→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

digitalio→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

digitalio→get_outputVoltage()

Retourne la source de tension utilisée pour piloter les bits en sortie.

digitalio→get_portDirection()

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

digitalio→get_portOpenDrain()

Retourne le type d'interface électrique de chaque bit du port (bitmap).

digitalio→get_portPolarity()

Retourne la polarité des bits du port (bitmap).

digitalio→get_portSize()

Retourne le nombre de bits implémentés dans le port d'E/S.

digitalio→get_portState()

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

digitalio→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

digitalio→isOnline()

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

digitalio→isOnline_async(callback, context)

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

digitalio→load(msValidity)

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

digitalio→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

digitalio→nextDigitalIO()

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

digitalio→pulse(bitno, ms_duration)

Déclenche une impulsion de durée spécifiée sur un bit choisi.

digitalio→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

digitalio→set_bitDirection(bitno, bitdirection)

Change la direction d'un seul bit du port d'E/S.

digitalio→set_bitOpenDrain(bitno, opendrain)

Change le type d'interface électrique d'un seul bit du port d'E/S.

digitalio→set_bitPolarity(bitno, bitpolarity)

Change la polarité d'un seul bit du port d'E/S.

3. Reference

digitalio→**set_bitState**(bitno, bitstate)

Change l'état d'un seul bit du port d'E/S.

digitalio→**set_logicalName**(newval)

Modifie le nom logique du port d'E/S digital.

digitalio→**set_outputVoltage**(newval)

Modifie la source de tension utilisée pour piloter les bits en sortie.

digitalio→**set_portDirection**(newval)

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

digitalio→**set_portOpenDrain**(newval)

Modifie le type d'interface électrique de chaque bit du port (bitmap).

digitalio→**set_portPolarity**(newval)

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

digitalio→**set_portState**(newval)

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

digitalio→**set_userData**(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`.

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( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port d'E/S digital soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDigitalIO.isOnline()` pour tester si le port d'E/S digital est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le port d'E/S digital sans ambiguïté

Retourne :

un objet de classe `YDigitalIO` qui permet ensuite de contrôler le port d'E/S digital.

YDigitalIO.FirstDigitalIO() yFirstDigitalIO()yFirstDigitalIO()

YDigitalIO

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

```
function yFirstDigitalIO( )
```

Utiliser la fonction `YDigitalIO.nextDigitalIO()` pour itérer sur les autres ports d'E/S digitaux.

Retourne :

un pointeur sur un objet `YDigitalIO`, correspondant au premier port d'E/S digital accessible en ligne, ou `null` si il n'y a pas de ports d'E/S digitaux disponibles.

digitalio→delayedPulse()digitalio.delayedPulse()**YDigitalIO**

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

```
function delayedPulse( bitno, ms_delay, ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

Paramètres :

- bitno** index du bit dans le port; le bit de poids faible est à l'index 0
- ms_delay** délai d'attente avant l'impulsion, en millisecondes
- ms_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**describe()****digitalio.describe()****YDigitalIO**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le port d'E/S digital (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

digitalio→**get_advertisedValue()****YDigitalIO****digitalio**→**advertisedValue()****digitalio.get_advertisedValue()**

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du port d'E/S digital (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

digitalio→**get_bitDirection()**

YDigitalIO

digitalio→**bitDirection()****digitalio.get_bitDirection()**

Retourne la direction d'un seul bit du port d'E/S.

```
function get_bitDirection( bitno)
```

(0 signifie que le bit est une entrée, 1 une sortie)

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_bitOpenDrain()**

YDigitalIO

digitalio→**bitOpenDrain()****digitalio.get_bitOpenDrain()**

Retourne la direction d'un seul bit du port d'E/S.

```
function get_bitOpenDrain( bitno)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert)..

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_bitPolarity()**

YDigitalIO

digitalio→**bitPolarity()****digitalio.get_bitPolarity()**

Retourne la polarité d'un seul bit du port d'E/S.

```
function get_bitPolarity( bitno)
```

0 signifie que l'entrée sortie est en mode normal, 1 qu'elle est en mode inverse

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_bitState()**

YDigitalIO

digitalio→**bitState()****digitalio.get_bitState()**

Retourne l'état d'un seul bit du port d'E/S.

```
function get_bitState( bitno)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

l'état du bit (0 ou 1).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_errorMessage()**

YDigitalIO

digitalio→**errorMessage()**

digitalio.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

digitalio→**get_errorType()****YDigitalIO****digitalio**→**errorType()****digitalio.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

digitalio→**get_friendlyName()**

YDigitalIO

digitalio→**friendlyName()****digitalio.get_friendlyName()**

Retourne un identifiant global du port d'E/S digital au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du port d'E/S digital si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port d'E/S digital (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le port d'E/S digital en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

digitalio→**get_functionDescriptor()****YDigitalIO****digitalio**→**functionDescriptor()****digitalio.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

digitalio→**get_functionId()**

YDigitalIO

digitalio→**functionId()****digitalio.get_functionId()**

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le port d'E/S digital (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

digitalio→**get_hardwareId()****YDigitalIO****digitalio**→**hardwareId()****digitalio.get_hardwareId()**

Retourne l'identifiant matériel unique du port d'E/S digital au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port d'E/S digital (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le port d'E/S digital (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

digitalio→**get_logicalName()**

YDigitalIO

digitalio→**logicalName()****digitalio.get_logicalName()**

Retourne le nom logique du port d'E/S digital.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du port d'E/S digital. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

digitalio→**get_module()****YDigitalIO****digitalio**→**module()****digitalio.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

digitalio→**get_module_async()**

YDigitalIO

digitalio→**module_async()**

digitalio.get_module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

digitalio→**get_outputVoltage()****YDigitalIO****digitalio**→**outputVoltage()****digitalio.get_outputVoltage()**

Retourne la source de tension utilisée pour piloter les bits en sortie.

```
function get_outputVoltage( )
```

Retourne :

une valeur parmi `Y_OUTPUTVOLTAGE_USB_5V`, `Y_OUTPUTVOLTAGE_USB_3V` et `Y_OUTPUTVOLTAGE_EXT_V` représentant la source de tension utilisée pour piloter les bits en sortie

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUTVOLTAGE_INVALID`.

digitalio→**get_portDirection()**

YDigitalIO

digitalio→**portDirection()****digitalio.get_portDirection()**

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
function get_portDirection( )
```

Retourne :

un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

En cas d'erreur, déclenche une exception ou retourne `Y_PORTDIRECTION_INVALID`.

digitalio→**get_portOpenDrain()****YDigitalIO****digitalio**→**portOpenDrain()****digitalio.get_portOpenDrain()**

Retourne le type d'interface électrique de chaque bit du port (bitmap).

```
function get_portOpenDrain( )
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

Retourne :

un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne `Y_PORTOPENDRAIN_INVALID`.

digitalio→**get_portPolarity()**

YDigitalIO

digitalio→**portPolarity()****digitalio.get_portPolarity()**

Retourne la polarité des bits du port (bitmap).

```
function get_portPolarity( )
```

Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

Retourne :

un entier représentant la polarité des bits du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne `Y_PORTPOLARITY_INVALID`.

digitalio→**get_portSize()****YDigitalIO****digitalio**→**portSize()****digitalio.get_portSize()**

Retourne le nombre de bits implémentés dans le port d'E/S.

```
function get_portSize( )
```

Retourne :

un entier représentant le nombre de bits implémentés dans le port d'E/S

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSIZE_INVALID`.

digitalio→**get_portState()**

YDigitalIO

digitalio→**portState()****digitalio.get_portState()**

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
function get_portState( )
```

Retourne :

un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSTATE_INVALID`.

digitalio→**get_userData()**

YDigitalIO

digitalio→**userData()****digitalio.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es 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.

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

function **isOnline**()

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le port d'E/S digital est joignable, `false` sinon

digitalio→**isOnline_async()**(digitalio.isOnline_async())**YDigitalIO**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**load_async()****digitalio.load_async()**

YDigitalIO

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

digitalio→**nextDigitalIO()****digitalio.nextDigitalIO()**

YDigitalIO

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

```
function nextDigitalIO( )
```

Retourne :

un pointeur sur un objet `YDigitalIO` accessible en ligne, ou `null` lorsque l'énumération est terminée.

digitalio→**pulse()****digitalio.pulse()**

YDigitalIO

Déclenche une impulsion de durée spécifiée sur un bit choisi.

```
function pulse( bitno, ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

Paramètres :

- bitno** index du bit dans le port; le bit de poids faible est à l'index 0
- ms_duration** durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**registerValueCallback()**
digitalio.registerValueCallback()

YDigitalIO

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

digitalio→**set_bitDirection()**

YDigitalIO

digitalio→**setBitDirection()****digitalio.set_bitDirection()**

Change la direction d'un seul bit du port d'E/S.

```
function set_bitDirection( bitno, bitdirection )
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

bitdirection nouvelle valeur de la direction, 0=entrée, 1=sortie. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_bitOpenDrain()**
digitalio→**setBitOpenDrain()**
digitalio.set_bitOpenDrain()

YDigitalIO

Change le type d'interface électrique d'un seul bit du port d'E/S.

```
function set_bitOpenDrain( bitno, opendrain)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0
opendrain 0 pour faire une entrée ou une sortie digitale standard, 1 pour une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_bitPolarity()**

YDigitalIO

digitalio→**setBitPolarity()****digitalio.set_bitPolarity()**

Change la polarité d'un seul bit du port d'E/S.

```
function set_bitPolarity( bitno, bitpolarity)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

bitpolarity nouvelle valeur de la polarité. 0=mode normal, 1=mode inverse. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_bitState()**

YDigitalIO

digitalio→**setBitState()****digitalio.set_bitState()**

Change l'état d'un seul bit du port d'E/S.

```
function set_bitState( bitno, bitstate)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

bitstate nouvel état du bit (1 ou 0)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_logicalName()****YDigitalIO****digitalio**→**setLogicalName()****digitalio.set_logicalName()**

Modifie le nom logique du port d'E/S digital.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du port d'E/S digital.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_outputVoltage()**
digitalio→**setOutputVoltage()**
digitalio.set_outputVoltage()

YDigitalIO

Modifie la source de tension utilisée pour piloter les bits en sortie.

```
function set_outputVoltage( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Paramètres :

newval une valeur parmi `Y_OUTPUTVOLTAGE_USB_5V`, `Y_OUTPUTVOLTAGE_USB_3V` et `Y_OUTPUTVOLTAGE_EXT_V` représentant la source de tension utilisée pour piloter les bits en sortie

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portDirection()**
digitalio→**setPortDirection()**
digitalio.set_portDirection()

YDigitalIO

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
function set_portDirection( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portOpenDrain()**
digitalio→**setPortOpenDrain()**
digitalio.set_portOpenDrain()

YDigitalIO

Modifie le type d'interface électrique de chaque bit du port (bitmap).

```
function set_portOpenDrain( newval)
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portPolarity()**

YDigitalIO

digitalio→**setPortPolarity()****digitalio.set_portPolarity()**

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

```
function set_portPolarity( newval)
```

Paramètres :

newval un entier représentant la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portState()**

YDigitalIO

digitalio→**setPortState()****digitalio.set_portState()**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
function set_portState( newval)
```

Seuls les bits configurés en sortie dans `portDirection` sont affectés.

Paramètres :

newval un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_userdata()**

YDigitalIO

digitalio→**setUserData()****digitalio.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( 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

Inverse l'état d'un seul bit du port d'E/S.

```
function toggle_bitState( bitno)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**wait_async(digitalio.wait_async())****YDigitalIO**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.13. Interface de la fonction Display

L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_display.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YDisplay = yoctolib.YDisplay;</code>
php	<code>require_once('yocto_display.php');</code>
cpp	<code>#include "yocto_display.h"</code>
m	<code>#import "yocto_display.h"</code>
pas	<code>uses yocto_display;</code>
vb	<code>yocto_display.vb</code>
cs	<code>yocto_display.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDisplay;</code>
py	<code>from yocto_display import *</code>

Fonction globales

yFindDisplay(func)

Permet de retrouver un écran 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 écran 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(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`.

display→**stopSequence(sequenceName)**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

display→**swapLayerContent(layerIdA, layerIdB)**

Permute le contenu de deux couches d'affichage.

display→**upload(pathname, content)**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

display→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDisplay.FindDisplay() yFindDisplay()yFindDisplay()

YDisplay

Permet de retrouver un ecran d'après un identifiant donné.

```
function yFindDisplay( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'ecran soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDisplay.isOnline()` pour tester si l'ecran 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'ecran sans ambiguïté

Retourne :

un objet de classe `YDisplay` qui permet ensuite de contrôler l'ecran.

YDisplay.FirstDisplay() yFirstDisplay()yFirstDisplay()

YDisplay

Commence l'énumération des écran accessibles par la librairie.

```
function yFirstDisplay( )
```

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écran.

Retourne :

un pointeur sur un objet `YDisplay`, correspondant au premier écran accessible en ligne, ou `null` si il n'y a pas de écran disponibles.

display→**copyLayerContent()**
display.copyLayerContent()**YDisplay**

Copie le contenu d'une couche d'affichage vers une autre couche.

```
function copyLayerContent( srcLayerId, dstLayerId)
```

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

srcLayerId l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

dstLayerId l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**describe()****display.describe()****YDisplay**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'ecran au format `TYPE (NAME) =SERIAL .FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'ecran (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

display→**fade()****display.fade()****YDisplay**

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

```
function fade( brightness, duration)
```

Paramètres :

brightness nouvelle valeur de luminosité de l'écran

duration durée en millisecondes de la transition.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**get_advertisedValue()**

YDisplay

display→**advertisedValue()**

display.get_advertisedValue()

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'écran (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

display→**get_brightness()****YDisplay****display**→**brightness()****display.get_brightness()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
function get_brightness( )
```

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne `Y_BRIGHTNESS_INVALID`.

display→**get_displayHeight()**

YDisplay

display→**displayHeight()****display.get_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

```
function get_displayHeight( )
```

Retourne :

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_DISPLAYHEIGHT_INVALID`.

display→**get_displayLayer()****YDisplay****display**→**displayLayer()****display.get_displayLayer()**

Retourne un objet `YDisplayLayer` utilisable pour dessiner sur la couche d'affichage correspondante.

```
function get_displayLayer( layerId)
```

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Paramètres :

layerId l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

Retourne :

un objet `YDisplayLayer`

En cas d'erreur, déclenche une exception ou retourne `null`.

display→**get_displayType()**

YDisplay

display→**displayType()****display.get_displayType()**

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

```
function get_displayType( )
```

Retourne :

une valeur parmi `Y_DISPLAYTYPE_MONO`, `Y_DISPLAYTYPE_GRAY` et `Y_DISPLAYTYPE_RGB` représentant le type de l'écran: monochrome, niveaux de gris ou couleur

En cas d'erreur, déclenche une exception ou retourne `Y_DISPLAYTYPE_INVALID`.

display→**get_displayWidth()****YDisplay****display**→**displayWidth()****display.get_displayWidth()**

Retourne la largeur de l'écran, en pixels.

```
function get_displayWidth( )
```

Retourne :

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_DISPLAYWIDTH_INVALID`.

display→**get_enabled()**

YDisplay

display→**enabled()****display.get_enabled()**

Retourne vrai si le l'ecran est alimenté, faux sinon.

```
function get_enabled( )
```

Retourne :

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon vrai si le l'ecran est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

display→**get_errorMessage()****YDisplay****display**→**errorMessage()****display**.**get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

display→**get_errorType()**

YDisplay

display→**errorType()****display.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

display→**get_friendlyName()****YDisplay****display**→**friendlyName()****display.get_friendlyName()**

Retourne un identifiant global de l'écran au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'écran si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'écran (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'écran en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

display→**get_functionDescriptor()**
display→**functionDescriptor()**
display.get_functionDescriptor()

YDisplay

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

display→**get_functionId()****YDisplay****display**→**functionId()****display.get_functionId()**

Retourne l'identifiant matériel de l'écran, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'écran (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

display→**get_hardwareId()**

YDisplay

display→**hardwareId()****display.get_hardwareId()**

Retourne l'identifiant matériel unique de l'ecran au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'ecran (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'ecran (ex: `RELAYLO1-123456.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( )
```

Retourne :

un entier représentant le nombre des couches affichables disponibles

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERCOUNT_INVALID`.

display→**get_layerHeight()**

YDisplay

display→**layerHeight()****display.get_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

```
function get_layerHeight( )
```

Retourne :

un entier représentant la hauteur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERHEIGHT_INVALID`.

display→**get_layerWidth()****YDisplay****display**→**layerWidth()****display.get_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

```
function get_layerWidth( )
```

Retourne :

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_LAYERWIDTH_INVALID`.

display→**get_logicalName()**

YDisplay

display→**logicalName()****display.get_logicalName()**

Retourne le nom logique de l'ecran.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'ecran. 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( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

display→**get_module_async()**

YDisplay

display→**module_async()**

display.get_module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

display→**get_orientation()****YDisplay****display**→**orientation()****display.get_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

```
function get_orientation( )
```

Retourne :

une valeur parmi `Y_ORIENTATION_LEFT`, `Y_ORIENTATION_UP`, `Y_ORIENTATION_RIGHT` et `Y_ORIENTATION_DOWN` représentant l'orientation sélectionnée pour l'écran

En cas d'erreur, déclenche une exception ou retourne `Y_ORIENTATION_INVALID`.

display→**get_startupSeq()**

YDisplay

display→**startupSeq()****display.get_startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

```
function get_startupSeq( )
```

Retourne :

une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

En cas d'erreur, déclenche une exception ou retourne `Y_STARTUPSEQ_INVALID`.

display→`get_userData()`**YDisplay****display**→`userData()``display.get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

display→**isOnline()****display.isOnline()**

YDisplay

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'écran sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'écran est joignable, `false` sinon

display→**isOnline_async()****display.isOnline_async()****YDisplay**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'écran sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

display→**load()****display.load()****YDisplay**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**load_async()****display.load_async()****YDisplay**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

display→**newSequence()****display.newSequence()**

YDisplay

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

```
function newSequence( )
```

Le nom de la séquence sera donné au moment de l'appel à `saveSequence()`, une fois la séquence terminée.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**nextDisplay()****display.nextDisplay()****YDisplay**

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

```
function nextDisplay( )
```

Retourne :

un pointeur sur un objet `YDisplay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

display→pauseSequence()display.pauseSequence()

YDisplay

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

```
function pauseSequence( delay_ms)
```

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

Paramètres :

delay_ms la durée de l'attente, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**playSequence()****display.playSequence()****YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

```
function playSequence( sequenceName)
```

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**registerValueCallback()**
display.registerValueCallback()**YDisplay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

display→**resetAll()****display.resetAll()****YDisplay**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

```
function resetAll( )
```

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→saveSequence()display.saveSequence()

YDisplay

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

```
function saveSequence( sequenceName)
```

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_brightness()****YDisplay****display**→**setBrightness()****display.set_brightness()**

Modifie la luminosité de l'écran.

```
function set_brightness( newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité 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_enabled()**

YDisplay

display→**setEnabled()****display.set_enabled()**

Modifie l'état d'activité de l'écran.

```
function set_enabled( newval)
```

Paramètres :

newval soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon l'état d'activité de l'écran

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_logicalName()****YDisplay****display**→**setLogicalName()****display.set_logicalName()**

Modifie le nom logique de l'écran.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'écran.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_orientation()**

YDisplay

display→**setOrientation()****display.set_orientation()**

Modifie l'orientation de l'écran.

```
function set_orientation( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_ORIENTATION_LEFT`, `Y_ORIENTATION_UP`, `Y_ORIENTATION_RIGHT` et `Y_ORIENTATION_DOWN` représentant l'orientation de l'écran

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_startupSeq()****YDisplay****display**→**setStartupSeq()****display.set_startupSeq()**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
function set_startupSeq( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_userData()**

YDisplay

display→**setUserData()****display.set_userData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
function set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

display→**stopSequence()****display.stopSequence()****YDisplay**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

```
function stopSequence( )
```

L'affichage est laissé tel quel.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**swapLayerContent()**
display.swapLayerContent()

YDisplay

Permute le contenu de deux couches d'affichage.

```
function swapLayerContent( layerIdA, layerIdB)
```

La couleur et la transparence de tous les pixels des deux couches sont 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.

```
function upload( pathname, content )
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→wait_async()display.wait_async()

YDisplay

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.14. Interface des objets DisplayLayer

Un DisplayLayer est une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_display.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDisplay = yoctolib.YDisplay;
php	require_once('yocto_display.php');
cpp	#include "yocto_display.h"
m	#import "yocto_display.h"
pas	uses yocto_display;
vb	yocto_display.vb
cs	yocto_display.cs
java	import com.yoctopuce.YoctoAPI.YDisplay;
py	from yocto_display import *

Méthodes des objets YDisplayLayer

displaylayer→clear()

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

displaylayer→clearConsole(text)

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

displaylayer→consoleOut(text)

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

displaylayer→drawBar(x1, y1, x2, y2)

Dessine un rectangle plein à une position spécifiée.

displaylayer→drawBitmap(x, y, w, bitmap, bgcolor)

Dessine un bitmap à la position spécifiée de la couche.

displaylayer→drawCircle(x, y, r)

Dessine un cercle vide à une position spécifiée.

displaylayer→drawDisc(x, y, r)

Dessine un disque plein à une position spécifiée.

displaylayer→drawImage(x, y, imagename)

Dessine une image GIF à la position spécifiée de la couche.

displaylayer→drawPixel(x, y)

Dessine un pixel unique à une position spécifiée.

displaylayer→drawRect(x1, y1, x2, y2)

Dessine un rectangle vide à une position spécifiée.

displaylayer→drawText(x, y, anchor, text)

Affiche un texte à la position spécifiée de la couche.

displaylayer→get_display()

Retourne l'YDisplay parent.

displaylayer→get_displayHeight()

Retourne la hauteur de l'écran, en pixels.

displaylayer→get_displayWidth()

Retourne la largeur de l'écran, en pixels.

3. Reference

displaylayer→**get_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

displaylayer→**get_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

displaylayer→**hide()**

Cache la couche de dessin.

displaylayer→**lineTo(x, y)**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

displaylayer→**moveTo(x, y)**

Déplace le point de dessin courant de cette couche à la position spécifiée.

displaylayer→**reset()**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

displaylayer→**selectColorPen(color)**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→**selectEraser()**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

displaylayer→**selectFont(fontname)**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

displaylayer→**selectGrayPen(graylevel)**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→**setAntialiasingMode(mode)**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

displaylayer→**setConsoleBackground(bgcol)**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

displaylayer→**setConsoleMargins(x1, y1, x2, y2)**

Configure les marges d'affichage pour la fonction `consoleOut`.

displaylayer→**setConsoleWordWrap(wordwrap)**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

displaylayer→**setLayerPosition(x, y, scrollTime)**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

displaylayer→**unhide()**

Affiche la couche.

displaylayer→clear()displaylayer.clear()**YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

```
function clear( )
```

Cette méthode ne change pas les réglages de la couche. Si vous désirez remettre la couche dans son état initial, utilisez plutôt la méthode `reset()`.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→clearConsole()
displaylayer.clearConsole()

YDisplayLayer

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

```
function clearConsole( )
```

Paramètres :

text le texte à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**consoleOut()**displaylayer.consoleOut()**YDisplayLayer**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

```
function consoleOut( text)
```

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

Paramètres :

text le message à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**drawBar()****displaylayer.drawBar()**

YDisplayLayer

Dessine un rectangle plein à une position spécifiée.

```
function drawBar( x1, y1, x2, y2)
```

Paramètres :

x1 la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

y1 la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

x2 la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

y2 la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→drawBitmap()
displaylayer.drawBitmap()****YDisplayLayer**

Dessine un bitmap à la position spécifiée de la couche.

```
function drawBitmap( x, y, w, bitmap, bgcol)
```

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour laisser les pixels inchangés

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**drawCircle()****displaylayer.drawCircle()**

YDisplayLayer

Dessine un cercle vide à une position spécifiée.

```
function drawCircle( x, y, r)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle
- r** le rayon du cercle, en pixels

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**drawDisc()**displaylayer.drawDisc()**YDisplayLayer**

Dessine un disque plein à une position spécifiée.

```
function drawDisc( x, y, r)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque
- r** le rayon du disque, en pixels

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**drawImage()**(displaylayer.drawImage())

YDisplayLayer

Dessine une image GIF à la position spécifiée de la couche.

```
function drawImage( x, y, imagename)
```

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image
- imagename** le nom du fichier GIF à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**drawPixel()****displaylayer.drawPixel()****YDisplayLayer**

Dessine un pixel unique à une position spécifiée.

```
function drawPixel( x, y )
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche
- y** la distance en pixels depuis le haut de la couche

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawRect()displaylayer.drawRect()

YDisplayLayer

Dessine un rectangle vide à une position spécifiée.

```
function drawRect( x1, y1, x2, y2)
```

Paramètres :

x1 la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle

y1 la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle

x2 la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle

y2 la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**drawText()**(displaylayer.drawText())**YDisplayLayer**

Affiche un texte à la position spécifiée de la couche.

```
function drawText( x, y, anchor, text)
```

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte
- y** la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte
- anchor** le point d'ancrage du texte, choisi parmi l'énumération Y_ALIGN: Y_ALIGN_TOP_LEFT, Y_ALIGN_CENTER_LEFT, Y_ALIGN_BASELINE_LEFT, Y_ALIGN_BOTTOM_LEFT, Y_ALIGN_TOP_CENTER, Y_ALIGN_CENTER, Y_ALIGN_BASELINE_CENTER, Y_ALIGN_BOTTOM_CENTER, Y_ALIGN_TOP_DECIMAL, Y_ALIGN_CENTER_DECIMAL, Y_ALIGN_BASELINE_DECIMAL, Y_ALIGN_BOTTOM_DECIMAL, Y_ALIGN_TOP_RIGHT, Y_ALIGN_CENTER_RIGHT, Y_ALIGN_BASELINE_RIGHT, Y_ALIGN_BOTTOM_RIGHT.
- text** le texte à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**get_display()**

YDisplayLayer

displaylayer→**display()****displaylayer.get_display()**

Retourne l'YDisplay parent.

```
function get_display( )
```

Retourne l'objet YDisplay parent du YDisplayLayer courant.

Retourne :

un objet YDisplay

displaylayer→**get_displayHeight()****YDisplayLayer****displaylayer**→**displayHeight()****displaylayer.get_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

```
function get_displayHeight( )
```

Retourne :

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne `Y_DISPLAYHEIGHT_INVALID`.

displaylayer→**get_displayWidth()**

YDisplayLayer

displaylayer→**displayWidth()**

displaylayer.get_displayWidth()

Retourne la largeur de l'écran, en pixels.

```
function get_displayWidth( )
```

Retourne :

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_DISPLAYWIDTH_INVALID.

displaylayer→**get_layerHeight()****YDisplayLayer****displaylayer**→**layerHeight()****displaylayer.get_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

```
function get_layerHeight( )
```

Retourne :

un entier représentant la hauteur des couches affichables, en pixels. En cas d'erreur, déclenche une exception ou retourne Y_LAYERHEIGHT_INVALID.

displaylayer→**get_layerWidth()**

YDisplayLayer

displaylayer→**layerWidth()**

displaylayer.get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

```
function get_layerWidth( )
```

Retourne :

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_LAYERWIDTH_INVALID.

displaylayer→**hide()****displaylayer.hide()****YDisplayLayer**

Cache la couche de dessin.

```
function hide( )
```

L'état de la couche est préservé, mais la couche ne sera plus plus affichés à l'écran jusqu'au prochain appel à `unhide()`. Le fait de cacher la couche améliore les performances de toutes les primitives d'affichage, car il évite de consacrer inutilement des cycles de calcul à afficher les états intermédiaires (technique de double-buffering).

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→lineTo()displaylayer.lineTo()

YDisplayLayer

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

```
function lineTo( x, y)
```

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au point final
- y** la distance en pixels depuis le haut de la couche jusqu'au point final

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**moveTo()****displaylayer.moveTo()****YDisplayLayer**

Déplace le point de dessin courant de cette couche à la position spécifiée.

```
function moveTo( x, y)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche de dessin
- y** la distance en pixels depuis le haut de la couche de dessin

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`displaylayer`→`reset()``displaylayer.reset()`

YDisplayLayer

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

```
function reset( )
```

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear()`.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**selectColorPen()**
displaylayer.selectColorPen()

YDisplayLayer

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

```
function selectColorPen( color)
```

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

Paramètres :

color la couleur RGB désirée (sous forme d'entier 24 bits)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**displaylayer→selectEraser()
displaylayer.selectEraser()**

YDisplayLayer

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de l'affichage de texte et de copie d'images bitmaps.

```
function selectEraser( )
```

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**selectFont()****displaylayer.selectFont()****YDisplayLayer**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

```
function selectFont( fontname)
```

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

Paramètres :

fontname le nom du fichier définissant la police de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**selectGrayPen()**
displaylayer.selectGrayPen()**YDisplayLayer**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

```
function selectGrayPen( graylevel)
```

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), toute valeur inférieure à 128 conduit à un point noir, et toute valeur supérieure ou égale à 128 devient un point lumineux.

Paramètres :

graylevel le niveau de gris désiré, de 0 à 255

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**setAntialiasingMode()**
displaylayer.setAntialiasingMode()

YDisplayLayer

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

```
function setAntialiasingMode( mode)
```

L'anti-aliasing est atténué la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

Paramètres :

mode true pour activer l'antialiasing, false pour le désactiver.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**setConsoleBackground()**
displaylayer.setConsoleBackground()

YDisplayLayer

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

```
function setConsoleBackground( bgcol)
```

Paramètres :

bgcol le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**setConsoleMargins()**
displaylayer.setConsoleMargins()

YDisplayLayer

Configure les marges d'affichage pour la fonction `consoleOut`.

```
function setConsoleMargins( x1, y1, x2, y2)
```

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche
- y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure
- x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite
- y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**setConsoleWordWrap()**
displaylayer.setConsoleWordWrap()

YDisplayLayer

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

```
function setConsoleWordWrap( wordwrap)
```

Paramètres :

wordwrap `true` pour retourner à la ligne entre les mots seulements, `false` pour retourner à l'extrême droite de chaque ligne.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**setLayerPosition()**
displaylayer.setLayerPosition()

YDisplayLayer

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

```
function setLayerPosition( x, y, scrollTime)
```

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

Paramètres :

- x** la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
- y** la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
- scrollTime** durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→unhide()displaylayer.unhide()

YDisplayLayer

Affiche la couche.

```
function unhide( )
```

Affiche a nouveau la couche après la command hide.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.15. Interface de contrôle de l'alimentation

La librairie de programmation Yoctopuce permet de contrôler la source d'alimentation qui doit être utilisée pour les fonctions du module consommant beaucoup de courant. Le module est par ailleurs capable de couper automatiquement l'alimentation externe lorsqu'il détecte que la tension a trop chuté (batterie épuisée).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_dualpower.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YDualPower = yoctolib.YDualPower;</code>
php	<code>require_once('yocto_dualpower.php');</code>
cpp	<code>#include "yocto_dualpower.h"</code>
m	<code>#import "yocto_dualpower.h"</code>
pas	<code>uses yocto_dualpower;</code>
vb	<code>yocto_dualpower.vb</code>
cs	<code>yocto_dualpower.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDualPower;</code>
py	<code>from yocto_dualpower import *</code>

Fonction globales

yFindDualPower(func)

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

yFirstDualPower()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

Méthodes des objets YDualPower

dualpower→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

dualpower→get_advertisedValue()

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

dualpower→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

dualpower→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

dualpower→get_extVoltage()

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

dualpower→get_friendlyName()

Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE . NOM_FONCTION`.

dualpower→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

dualpower→get_functionId()

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

dualpower→get_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'alimentation au format `SERIAL . FUNCTIONID`.

dualpower→get_logicalName()

3. Reference

Retourne le nom logique du contrôle d'alimentation.

dualpower→**get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

dualpower→**get_module_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

dualpower→**get_powerControl()**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

dualpower→**get_powerState()**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

dualpower→**get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

dualpower→**isOnline()**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

dualpower→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

dualpower→**load(msValidity)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

dualpower→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

dualpower→**nextDualPower()**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

dualpower→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

dualpower→**set_logicalName(newval)**

Modifie le nom logique du contrôle d'alimentation.

dualpower→**set_powerControl(newval)**

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

dualpower→**set_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

dualpower→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDualPower.FindDualPower() yFindDualPower()yFindDualPower()

YDualPower

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

```
function yFindDualPower( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'alimentation soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDualPower.isOnline()` pour tester si le contrôle d'alimentation est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le contrôle d'alimentation sans ambiguïté

Retourne :

un objet de classe `YDualPower` qui permet ensuite de contrôler le contrôle d'alimentation.

YDualPower.FirstDualPower() yFirstDualPower()yFirstDualPower()

YDualPower

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

```
function yFirstDualPower( )
```

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

Retourne :

un pointeur sur un objet `YDualPower`, correspondant au premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

dualpower→describe()dualpower.describe()**YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le contrôle d'alimentation (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

dualpower→**get_advertisedValue()**

YDualPower

dualpower→**advertisedValue()**

dualpower.get_advertisedValue()

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du contrôle d'alimentation (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

dualpower→**get_errorMessage()****YDualPower****dualpower**→**errorMessage()****dualpower.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

dualpower→**get_errorType()**

YDualPower

dualpower→**errorType()****dualpower.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

dualpower→**get_extVoltage()****YDualPower****dualpower**→**extVoltage()****dualpower.get_extVoltage()**

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

```
function get_extVoltage( )
```

Retourne :

un entier représentant la tension mesurée sur l'alimentation de puissance externe, en millivolts

En cas d'erreur, déclenche une exception ou retourne `Y_EXTVOLTAGE_INVALID`.

dualpower→get_friendlyName()

YDualPower

dualpower→friendlyName()

dualpower.get_friendlyName()

Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du contrôle d'alimentation si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'alimentation (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

dualpower→**get_functionDescriptor()**
dualpower→**functionDescriptor()**
dualpower.get_functionDescriptor()

YDualPower

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur 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( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

dualpower→**get_hardwareId()****YDualPower****dualpower**→**hardwareId()****dualpower.get_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'alimentation au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'alimentation (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

dualpower→**get_logicalName()**

YDualPower

dualpower→**logicalName()**

dualpower.get_logicalName()

Retourne le nom logique du contrôle d'alimentation.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du contrôle d'alimentation. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

dualpower→**get_module()****YDualPower****dualpower**→**module()****dualpower.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

dualpower→**get_module_async()****YDualPower****dualpower**→**module_async()****dualpower.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

dualpower→**get_powerControl()****YDualPower****dualpower**→**powerControl()****dualpower.get_powerControl()**

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
function get_powerControl( )
```

Retourne :

une valeur parmi `Y_POWERCONTROL_AUTO`, `Y_POWERCONTROL_FROM_USB`, `Y_POWERCONTROL_FROM_EXT` et `Y_POWERCONTROL_OFF` représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne `Y_POWERCONTROL_INVALID`.

dualpower→**get_powerState()**

YDualPower

dualpower→**powerState()**

dualpower.get_powerState()

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

```
function get_powerState( )
```

Retourne :

une valeur parmi `Y_POWERSTATE_OFF`, `Y_POWERSTATE_FROM_USB` et `Y_POWERSTATE_FROM_EXT` représentant la source d'alimentation active pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne `Y_POWERSTATE_INVALID`.

dualpower→**get_userdata()****YDualPower****dualpower**→**userData()****dualpower.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es 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.

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le contrôle d'alimentation est joignable, `false` sinon

dualpower→**isOnline_async()**
dualpower.isOnline_async()**YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

dualpower→**load()****dualpower.load()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→**load_async()****dualpower.load_async()****YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

dualpower→**nextDualPower()**
dualpower.nextDualPower()

YDualPower

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

```
function nextDualPower( )
```

Retourne :

un pointeur sur un objet `YDualPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

dualpower→**registerValueCallback()**
dualpower.registerValueCallback()

YDualPower

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

dualpower→**set_logicalName()**
dualpower→**setLogicalName()**
dualpower.set_logicalName()

YDualPower

Modifie le nom logique du contrôle d'alimentation.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→**set_powerControl()**
dualpower→**setPowerControl()**
dualpower.set_powerControl()

YDualPower

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
function set_powerControl( newval)
```

Paramètres :

newval une valeur parmi `Y_POWERCONTROL_AUTO`, `Y_POWERCONTROL_FROM_USB`, `Y_POWERCONTROL_FROM_EXT` et `Y_POWERCONTROL_OFF` représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→**set_userdata()**

YDualPower

dualpower→**setUserData()****dualpower.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

dualpower→**wait_async()****dualpower.wait_async()****YDualPower**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.16. Interface de la fonction Files

L'interface de stockage de fichiers permet de stocker des fichiers sur certains modules, par exemple pour personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_files.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YFiles = yoctolib.YFiles;</code>
php	<code>require_once('yocto_files.php');</code>
c++	<code>#include "yocto_files.h"</code>
m	<code>#import "yocto_files.h"</code>
pas	<code>uses yocto_files;</code>
vb	<code>yocto_files.vb</code>
cs	<code>yocto_files.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YFiles;</code>
py	<code>from yocto_files import *</code>

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 filesystem et retourne son contenu.

files→download_async(pathname, callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

files→format_fs()

Rétabli le système de fichier dans on état original, défragmenté.

files→get_advertisedValue()

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

files→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

files→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

files→get_filesCount()

Retourne le nombre de fichiers présents dans le système de fichier.

files→get_freeSpace()

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

files→get_friendlyName()

Retourne un identifiant global du système de fichier au format `NOM_MODULE . NOM_FONCTION`.

files→get_functionDescriptor()

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(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.

files→**upload(pathname, content)**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

files→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YFiles.FindFiles() yFindFiles()yFindFiles()

YFiles

Permet de retrouver un système de fichier d'après un identifiant donné.

```
function yFindFiles( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le système de fichier sans ambiguïté

Retourne :

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

YFiles.FirstFiles()
yFirstFiles()yFirstFiles()

YFiles

Commence l'énumération des système de fichier accessibles par la librairie.

```
function yFirstFiles( )
```

Utiliser la fonction `YFiles.nextFiles()` pour itérer sur les autres système de fichier.

Retourne :

un pointeur sur un objet `YFiles`, correspondant au premier système de fichier accessible en ligne, ou `null` si il n'y a pas de système de fichier disponibles.

files→**describe()****files.describe()****YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le système de fichier (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

files→**download()****files.download()****YFiles**

Télécharge le fichier choisi du filesystem et retourne son contenu.

```
function download( pathname)
```

Paramètres :

pathname nom complet du fichier à charger, y compris le chemin d'accès.

Retourne :

le contenu du fichier chargé sous forme d'objet binaire

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

files→download_async(files.download_async())

YFiles

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
function download_async( pathname, callback, context)
```

Paramètres :

pathname nom complet du fichier à charger, y compris le chemin d'accès.

callback fonction fournie par l'utilisateur, qui sera appelée lorsque la suite du chargement aura été effectué. La fonction callback doit prendre trois arguments: - la variable de contexte à disposition de l'utilisateur - l'objet YFiles dont la méthode download_async a été appelée - le contenu du fichier chargé sous forme d'objet binaire

context variable de contexte à disposition de l'utilisateur

Retourne :

rien.

files→**format_fs()****files.format_fs()****YFiles**

Rétabli le système de fichier dans on état original, défragmenté.

```
function format_fs( )
```

entièrement vide. Tous les fichiers précédemment chargés sont irrémédiablement effacés.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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( )
```

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→**get_errorMessage()****YFiles****files**→**errorMessage()****files.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

files→**get_errorType()**

YFiles

files→**errorType()****files.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

files→**get_filesCount()****YFiles****files**→**filesCount()****files.get_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

```
function get_filesCount( )
```

Retourne :

un entier représentant le nombre de fichiers présents dans le système de fichier

En cas d'erreur, déclenche une exception ou retourne `Y_FILESCOUNT_INVALID`.

files→**get_freeSpace()**

YFiles

files→**freeSpace()****files.get_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

```
function get_freeSpace( )
```

Retourne :

un entier représentant l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets

En cas d'erreur, déclenche une exception ou retourne `Y_FREESPACE_INVALID`.

files→**get_friendlyName()****YFiles****files**→**friendlyName()****files.get_friendlyName()**

Retourne un identifiant global du système de fichier au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du système de fichier si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du système de fichier (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le système de fichier en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

files→**get_functionDescriptor()**

YFiles

files→**functionDescriptor()**

files.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

files→**get_functionId()****YFiles****files**→**functionId()****files.get_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le système de fichier (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

files→**get_hardwareId()**

YFiles

files→**hardwareId()****files.get_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le système de fichier (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

files→**get_list()****YFiles****files**→**list()****files.get_list()**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
function get_list( pattern)
```

Paramètres :

pattern un filtre optionel sur les noms de fichiers retournés, pouvant contenir des astérisques et des points d'interrogations comme jokers. Si le pattern fourni est vide, tous les fichiers sont retournés.

Retourne :

une liste d'objets YFileRecord, contenant le nom complet (y compris le chemin d'accès), la taille en octets et le CRC 32-bit du contenu du fichier.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

files→**get_logicalName()**

YFiles

files→**logicalName()****files.get_logicalName()**

Retourne le nom logique du système de fichier.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du système de fichier. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

files→**get_module()****YFiles****files**→**module()****files.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

files→**get_module_async()****files**→**module_async()****files.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

files→**get_userData()****YFiles****files**→**userData()****files.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

files→isOnline()files.isOnline()

YFiles

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le système de fichier est joignable, `false` sinon

files→**isOnline_async()****files.isOnline_async()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

files→**load()****files.load()**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**load_async()****files.load_async()****YFiles**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

`files`→`nextFiles()``files.nextFiles()`

YFiles

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles()`.

```
function nextFiles( )
```

Retourne :

un pointeur sur un objet `YFiles` accessible en ligne, ou `null` lorsque l'énumération est terminée.

files→**registerValueCallback()**
files.registerValueCallback()

YFiles

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

files→**remove()****files.remove()**

Efface un fichier, spécifié par son path complet, du système de fichier.

```
function remove( pathname)
```

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occupe. Par contre, la ré-écriture d'un fichier du même nom récupérera dans tout les cas l'espace qui n'aurait éventuellement pas été libéré. Pour s'assurer de libérer la totalité de l'espace du système de fichier, utilisez la fonction `format_fs`.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**set_logicalName()****YFiles****files**→**setLogicalName()****files.set_logicalName()**

Modifie le nom logique du système de fichier.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du système de fichier.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**set_userdata()**

YFiles

files→**setUserData()****files.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

files→**upload()****files.upload()**

YFiles

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

```
function upload( pathname, content)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→wait_async()files.wait_async()

YFiles

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.17. Interface de la fonction GenericSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_genericsensor.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YGenericSensor = yoctolib.YGenericSensor;
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_FUNCTION`.

genericsensor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

genericsensor→get_functionId()

Retourne l'identifiant matériel du capteur générique, sans référence au module.

genericsensor→get_hardwareId()

Retourne l'identifiant matériel unique du capteur générique au format SERIAL . FUNCTIONID.

genericsensor→**get_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

genericsensor→**get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

genericsensor→**get_logicalName()**

Retourne le nom logique du capteur générique.

genericsensor→**get_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

genericsensor→**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

genericsensor→**get_module_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

genericsensor→**get_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

genericsensor→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

genericsensor→**get_resolution()**

Retourne la résolution des valeurs mesurées.

genericsensor→**get_signalRange()**

Retourne la plage de signal électrique utilisée par le capteur.

genericsensor→**get_signalUnit()**

Retourne l'unité du signal électrique utilisée par le capteur.

genericsensor→**get_signalValue()**

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

genericsensor→**get_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

genericsensor→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

genericsensor→**get_valueRange()**

Retourne la plage de valeurs physiques mesurés par le capteur.

genericsensor→**isOnline()**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

genericsensor→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

genericsensor→**load(msValidity)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

genericsensor→**loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

genericsensor→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

genericsensor→**nextGenericSensor()**

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

`genericsensor`→`registerTimedReportCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

`genericsensor`→`registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

`genericsensor`→`set_highestValue(newval)`

Modifie la mémoire de valeur maximale observée.

`genericsensor`→`set_logFrequency(newval)`

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

`genericsensor`→`set_logicalName(newval)`

Modifie le nom logique du capteur générique.

`genericsensor`→`set_lowestValue(newval)`

Modifie la mémoire de valeur minimale observée.

`genericsensor`→`set_reportFrequency(newval)`

Modifie la fréquence de notification périodique des valeurs mesurées.

`genericsensor`→`set_resolution(newval)`

Modifie la résolution des valeurs physique mesurées.

`genericsensor`→`set_signalRange(newval)`

Modifie la plage de signal électrique utilisée par le capteur.

`genericsensor`→`set_unit(newval)`

Change l'unité dans laquelle la valeur mesurée est exprimée.

`genericsensor`→`set_userData(data)`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

`genericsensor`→`set_valueRange(newval)`

Modifie la plage de valeurs physiques mesurés par le capteur.

`genericsensor`→`wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YGenericSensor.FindGenericSensor() yFindGenericSensor()yFindGenericSensor()

YGenericSensor

Permet de retrouver un capteur générique d'après un identifiant donné.

```
function yFindGenericSensor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur générique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGenericSensor.isOnline()` pour tester si le capteur générique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur générique sans ambiguïté

Retourne :

un objet de classe `YGenericSensor` qui permet ensuite de contrôler le capteur générique.

**YGenericSensor.FirstGenericSensor()
yFirstGenericSensor()yFirstGenericSensor()**

YGenericSensor

Commence l'énumération des capteurs génériques accessibles par la librairie.

```
function yFirstGenericSensor( )
```

Utiliser la fonction `YGenericSensor.nextGenericSensor()` pour itérer sur les autres capteurs génériques.

Retourne :

un pointeur sur un objet `YGenericSensor`, correspondant au premier capteur générique accessible en ligne, ou `null` si il n'y a pas de capteurs génériques disponibles.

genericsensor→**calibrateFromPoints()**
genericsensor.calibrateFromPoints()**YGenericSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→describe()genericsensor.describe()**YGenericSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format `TYPE (NAME) =SERIAL .FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur générique (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

genericsensor→**get_advertisedValue()**

YGenericSensor

genericsensor→**advertisedValue()**

genericsensor.get_advertisedValue()

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur générique (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

genericsensor→**get_currentRawValue()****YGenericSensor****genericsensor**→**currentRawValue()****genericsensor.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

genericsensor→**get_currentValue()**
genericsensor→**currentValue()**
genericsensor.get_currentValue()

YGenericSensor

Retourne la valeur mesurée actuelle.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

genericsensor→get_errorMessage()
genericsensor→errorMessage()
genericsensor.get_errorMessage()

YGenericSensor

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

genericsensor→get_errorType()

YGenericSensor

genericsensor→errorType()

genericsensor.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

genericsensor→**get_friendlyName()****YGenericSensor****genericsensor**→**friendlyName()****genericsensor.get_friendlyName()**

Retourne un identifiant global du capteur générique au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur générique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur générique (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur générique en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

genericsensor→**get_functionDescriptor()**
genericsensor→**functionDescriptor()**
genericsensor.get_functionDescriptor()

YGenericSensor

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

genericsensor→**get_functionId()**
genericsensor→**functionId()**
genericsensor.get_functionId()

YGenericSensor

Retourne l'identifiant matériel du capteur générique, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur générique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

genericsensor→**get_hardwareId()**
genericsensor→**hardwareId()**
genericsensor.get_hardwareId()

YGenericSensor

Retourne l'identifiant matériel unique du capteur générique au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur générique (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur générique (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

genericsensor→**get_highestValue()****YGenericSensor****genericsensor**→**highestValue()****genericsensor.get_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

genericsensor→**get_logFrequency()**

YGenericSensor

genericsensor→**logFrequency()**

genericsensor.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

genericsensor→**get_logicalName()****YGenericSensor****genericsensor**→**logicalName()****genericsensor.get_logicalName()**

Retourne le nom logique du capteur générique.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur générique. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

genericsensor→**get_lowestValue()**

YGenericSensor

genericsensor→**lowestValue()**

genericsensor.get_lowestValue()

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

genericsensor→**get_module()**
genericsensor→**module()**
genericsensor.get_module()

YGenericSensor

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

genericsensor→**get_module_async()**
genericsensor→**module_async()**
genericsensor.get_module_async()

YGenericSensor

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

genericsensor→**get_recordedData()****YGenericSensor****genericsensor**→**recordedData()****genericsensor.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

genericsensor→get_reportFrequency()

YGenericSensor

genericsensor→reportFrequency()

genericsensor.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

genericsensor→**get_resolution()****YGenericSensor****genericsensor**→**resolution()****genericsensor.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

genericsensor→get_signalRange()

YGenericSensor

genericsensor→signalRange()

genericsensor.get_signalRange()

Retourne la plage de signal électrique utilisée par le capteur.

```
function get_signalRange( )
```

Retourne :

une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y_SIGNALRANGE_INVALID.

genericsensor→**get_signalUnit()****YGenericSensor****genericsensor**→**signalUnit()****genericsensor.get_signalUnit()**

Retourne l'unité du signal électrique utilisée par le capteur.

```
function get_signalUnit( )
```

Retourne :

une chaîne de caractères représentant l'unité du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALUNIT_INVALID`.

genericsensor→**get_signalValue()**
genericsensor→**signalValue()**
genericsensor.get_signalValue()

YGenericSensor

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

```
function get_signalValue( )
```

Retourne :

une valeur numérique représentant la valeur mesurée du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALVALUE_INVALID`.

genericsensor→**get_unit()****YGenericSensor****genericsensor**→**unit()****genericsensor.get_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

genericsensor→get_userData()

YGenericSensor

genericsensor→userData()

genericsensor.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

```
function get_userData( )
```

Cet attribut n'es 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()**YGenericSensor****genericsensor→valueRange()****genericsensor.get_valueRange()**

Retourne la plage de valeurs physiques mesurés par le capteur.

```
function get_valueRange( )
```

Retourne :

une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

En cas d'erreur, déclenche une exception ou retourne Y_VALUERANGE_INVALID.

genericsensor→**isOnline()****genericsensor.isOnline()**

YGenericSensor

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur générique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur générique est joignable, `false` sinon

genericsensor→**isOnline_async()**
genericsensor.isOnline_async()**YGenericSensor**

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur générique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

genericsensor→**load()****genericsensor.load()****YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**loadCalibrationPoints()**
genericsensor.loadCalibrationPoints()**YGenericSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**load_async()**
genericsensor.load_async()**YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

genericSensor→**nextGenericSensor()**
genericSensor.nextGenericSensor()

YGenericSensor

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

```
function nextGenericSensor( )
```

Retourne :

un pointeur sur un objet `YGenericSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

genericsensor→**registerTimedReportCallback()**
genericsensor.registerTimedReportCallback()

YGenericSensor

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

genericsensor→**set_highestValue()**
genericsensor→**setHighestValue()**
genericsensor.set_highestValue()

YGenericSensor

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericSensor→**set_logFrequency()**
genericSensor→**setLogFrequency()**
genericSensor.set_logFrequency()

YGenericSensor

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_logicalName()**

YGenericSensor

genericsensor→**setLogicalName()**

genericsensor.set_logicalName()

Modifie le nom logique du capteur générique.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur générique.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_lowestValue()
genericsensor→setLowestValue()
genericsensor.set_lowestValue()

YGenericSensor

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_reportFrequency()**
genericsensor→**setReportFrequency()**
genericsensor.set_reportFrequency()

YGenericSensor

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_resolution()**
genericsensor→**setResolution()**
genericsensor.set_resolution()

YGenericSensor

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_signalRange()**

YGenericSensor

genericsensor→**setSignalRange()**

genericsensor.set_signalRange()

Modifie la plage de signal électrique utilisée par le capteur.

```
function set_signalRange( newval)
```

Paramètres :

newval une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_unit()****YGenericSensor****genericsensor**→**setUnit()****genericsensor.set_unit()**

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
function set_unit( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_userdata()**
genericsensor→**setUserData()**
genericsensor.set_userdata()

YGenericSensor

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

genericSensor→**set_valueRange()**
genericSensor→**setValueRange()**
genericSensor.set_valueRange()

YGenericSensor

Modifie la plage de valeurs physiques mesurés par le capteur.

```
function set_valueRange( newval)
```

Le changement de plage peut avoir pour effet de bord un changement automatique de la résolution affichée.

Paramètres :

newval une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**wait_async()** **genericsensor.wait_async()**

YGenericSensor

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.18. Interface de la fonction Gyro

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_gyro.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YGyro = yoctolib.YGyro;
php	require_once('yocto_gyro.php');
cpp	#include "yocto_gyro.h"
m	#import "yocto_gyro.h"
pas	uses yocto_gyro;
vb	yocto_gyro.vb
cs	yocto_gyro.cs
java	import com.yoctopuce.YoctoAPI.YGyro;
py	from yocto_gyro import *

Fonction globales

yFindGyro(func)

Permet de retrouver un gyroscope d'après un identifiant donné.

yFirstGyro()

Commence l'énumération des gyroscopes accessibles par la librairie.

Méthodes des objets YGyro

gyro→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

gyro→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

gyro→get_advertisedValue()

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

gyro→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

gyro→get_currentValue()

Retourne la valeur actuelle de la vitesse angulaire.

gyro→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

gyro→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

gyro→get_friendlyName()

Retourne un identifiant global du gyroscope au format `NOM_MODULE . NOM_FONCTION`.

gyro→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

gyro→get_functionId()

Retourne l'identifiant matériel du gyroscope, sans référence au module.

gyro→get_hardwareId()

Retourne l'identifiant matériel unique du gyroscope au format `SERIAL . FUNCTIONID`.

gyro→**get_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

gyro→**get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

gyro→**get_logicalName()**

Retourne le nom logique du gyroscope.

gyro→**get_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

gyro→**get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

gyro→**get_module_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

gyro→**get_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_quaternionX()**

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_quaternionY()**

Retourne la composante y du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_quaternionZ()**

Retourne la composante z du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_recordedData(startTime, endTime)**

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

gyro→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

gyro→**get_resolution()**

Retourne la résolution des valeurs mesurées.

gyro→**get_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→**get_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

gyro→**get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

gyro→**get_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

gyro→get_yValue()

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

gyro→get_zValue()

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

gyro→isOnline()

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

gyro→isOnline_async(callback, context)

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

gyro→load(msValidity)

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

gyro→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

gyro→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

gyro→nextGyro()

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

gyro→registerAnglesCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

gyro→registerQuaternionCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

gyro→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

gyro→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

gyro→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

gyro→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

gyro→set_logicalName(newval)

Modifie le nom logique du gyroscope.

gyro→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

gyro→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

gyro→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

gyro→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

gyro→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YGyro.FindGyro() yFindGyro()yFindGyro()

YGyro

Permet de retrouver un gyroscope d'après un identifiant donné.

```
function yFindGyro( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le gyroscope sans ambiguïté

Retourne :

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

**YGyro.FirstGyro()
yFirstGyro()yFirstGyro()**

YGyro

Commence l'énumération des gyroscopes accessibles par la librairie.

```
function yFirstGyro( )
```

Utiliser la fonction `YGyro.nextGyro()` pour itérer sur les autres gyroscopes.

Retourne :

un pointeur sur un objet `YGyro`, correspondant au premier gyroscope accessible en ligne, ou `null` si il n'y a pas de gyroscopes disponibles.

gyro→**calibrateFromPoints()**
gyro.calibrateFromPoints()**YGyro**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→describe()gyro.describe()**YGyro**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le gyroscope (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

gyro→**get_advertisedValue()**

YGyro

gyro→**advertisedValue()****gyro.get_advertisedValue()**

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du gyroscope (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

gyro→**get_currentRawValue()****YGyro****gyro**→**currentRawValue()****gyro.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

gyro→**get_currentValue()**

YGyro

gyro→**currentValue()****gyro.get_currentValue()**

Retourne la valeur actuelle de la vitesse angulaire.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de la vitesse angulaire

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

gyro→**get_errorMessage()****YGyro****gyro**→**errorMessage()****gyro.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

gyro→**get_errorType()**

YGyro

gyro→**errorType()****gyro.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

gyro→**get_friendlyName()****YGyro****gyro**→**friendlyName()****gyro.get_friendlyName()**

Retourne un identifiant global du gyroscope au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du gyroscope si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du gyroscope (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le gyroscope en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

gyro→**get_functionDescriptor()**
gyro→**functionDescriptor()**
gyro.get_functionDescriptor()

YGyro

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

gyro→**get_functionId()****YGyro****gyro**→**functionId()****gyro.get_functionId()**

Retourne l'identifiant matériel du gyroscope, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le gyroscope (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

gyro→**get_hardwareId()**

YGyro

gyro→**hardwareId()****gyro.get_hardwareId()**

Retourne l'identifiant matériel unique du gyroscope au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le gyroscope (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

gyro→**get_heading()****YGyro****gyro**→**heading()****gyro.get_heading()**

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_heading( )
```

L'axe de lacet peut être attribué à n'importe laquelle des directions 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( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

gyro→**get_logFrequency()****YGyro****gyro**→**logFrequency()****gyro.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

gyro→**get_logicalName()**

YGyro

gyro→**logicalName()****gyro.get_logicalName()**

Retourne le nom logique du gyroscope.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du gyroscope. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

gyro→**get_lowestValue()****YGyro****gyro**→**lowestValue()****gyro.get_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

gyro→**get_module()**

YGyro

gyro→**module()****gyro.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

gyro→**get_module_async()****YGyro****gyro**→**module_async()****gyro.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

gyro→**get_pitch()**

YGyro

gyro→**pitch()****gyro.get_pitch()**

Retourne une estimation de l'assiette (angle de tangage), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_pitch( )
```

L'axe de tangage peut être attribué à n'importe laquelle des directions physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :

un nombre à virgule correspondant à l'assiette, exprimée en degrés (entre -90 et +90).

gyro→**get_quaternionW()****YGyro****gyro**→**quaternionW()****gyro.get_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_quaternionW( )
```

Retourne :

un nombre à virgule correspondant à la composante w du quaternion.

gyro→**get_quaternionX()**

YGyro

gyro→**quaternionX()****gyro.get_quaternionX()**

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_quaternionX( )
```

La composante x est essentiellement corrélée aux rotations sur l'axe de roulis.

Retourne :

un nombre à virgule correspondant à la composante x du quaternion.

gyro→**get_quaternionY()****YGyro****gyro**→**quaternionY()****gyro.get_quaternionY()**

Retourne la composante y du quaternion décrivant l'orientation estimatée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_quaternionY( )
```

La composante y est essentiellement corrélée aux rotations sur l'axe de tangage.

Retourne :

un nombre à virgule correspondant à la composante y du quaternion.

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( )
```

La composante z est essentiellement corrélée aux rotations sur l'axe de lacet.

Retourne :

un nombre à virgule correspondant à la composante z du quaternion.

gyro→**get_recordedData()****YGyro****gyro**→**recordedData()****gyro.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

gyro→**get_reportFrequency()**

YGyro

gyro→**reportFrequency()****gyro.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

gyro→**get_resolution()**
gyro→**resolution()****gyro.get_resolution()**

YGyro

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

gyro→**get_roll()**

YGyro

gyro→**roll()****gyro.get_roll()**

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
function get_roll( )
```

L'axe de roulis peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :

un nombre à virgule correspondant à l'inclinaison, exprimée en degrés (entre -180 et +180).

gyro→**get_unit()****YGyro****gyro**→**unit()****gyro.get_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la vitesse angulaire est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

gyro→**get_userData()**

YGyro

gyro→**userData()****gyro.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

gyro→**get_xValue()****YGyro****gyro**→**xValue()****gyro.get_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

```
function get_xValue( )
```

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_XVALUE_INVALID`.

gyro→**get_yValue()**

YGyro

gyro→**yValue()****gyro.get_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

```
function get_yValue( )
```

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_YVALUE_INVALID`.

gyro→**get_zValue()****YGyro****gyro**→**zValue()****gyro.get_zValue()**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

```
function get_zValue( )
```

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_ZVALUE_INVALID`.

gyro→isOnline()gyro.isOnline()

YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

fonction **isOnline**()

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le gyroscope est joignable, `false` sinon

gyro→isOnline_async()gyro.isOnline_async()

YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

gyro→**load()****gyro.load()****YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→loadCalibrationPoints()
gyro.loadCalibrationPoints()**YGyro**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**load_async()****gyro.load_async()****YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

gyro→**nextGyro()****gyro.nextGyro()****YGyro**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

```
function nextGyro( )
```

Retourne :

un pointeur sur un objet `YGyro` accessible en ligne, ou `null` lorsque l'énumération est terminée.

gyro→**registerAnglesCallback()**
gyro.registerAnglesCallback()**YGyro**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
function registerAnglesCallback( callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelés trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter quatre arguments: l'objet YGyro du module qui a tourné, et les valeurs des trois angles roll, pitch et heading en degrés (nombres à virgules).

gyro→registerQuaternionCallback()
gyro.registerQuaternionCallback()

YGyro

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
function registerQuaternionCallback( callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelés trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter cinq arguments: l'objet YGyro du module qui a tourné, et les valeurs des quatre composantes w, x, y et z du quaternion (nombres à virgules).

gyro→registerTimedReportCallback()
gyro.registerTimedReportCallback()**YGyro**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

gyro→registerValueCallback()
gyro.registerValueCallback()

YGyro

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

gyro→**set_highestValue()**

YGyro

gyro→**setHighestValue()****gyro.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→set_logFrequency()**YGyro****gyro→setLogFrequency()gyro.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_logicalName()**

YGyro

gyro→**setLogicalName()****gyro.set_logicalName()**

Modifie le nom logique du gyroscope.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du gyroscope.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_lowestValue()****YGyro****gyro**→**setLowestValue()****gyro.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→set_reportFrequency()
gyro→setReportFrequency()
gyro.set_reportFrequency()

YGyro

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→set_resolution()**YGyro****gyro→setResolution()gyro.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_userdata()**

YGyro

gyro→**setUserData()****gyro.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

gyro→**wait_async()****gyro.wait_async()****YGyro**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.19. Interface d'un port de Yocto-hub

Les objets YHubPort permettent de contrôler l'alimentation des ports d'un YoctoHub, ainsi que de détecter si un module y est raccordé et lequel. Un YHubPort reçoit toujours automatiquement comme nom logique le numéro de série unique du module Yoctopuce qui y est connecté.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_hubport.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YHubPort = yoctolib.YHubPort;
php	require_once('yocto_hubport.php');
c++	#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( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port de Yocto-hub soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHubPort.isOnline()` pour tester si le port de Yocto-hub est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le port de Yocto-hub sans ambiguïté

Retourne :

un objet de classe `YHubPort` qui permet ensuite de contrôler le port de Yocto-hub.

YHubPort.FirstHubPort()
yFirstHubPort()yFirstHubPort()

YHubPort

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

```
function yFirstHubPort( )
```

Utiliser la fonction `YHubPort.nextHubPort()` pour itérer sur les autres port de Yocto-hub.

Retourne :

un pointeur sur un objet `YHubPort`, correspondant au premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

hubport→**describe()****hubport.describe()****YHubPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le port de Yocto-hub (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

hubport→**get_advertisedValue()****YHubPort****hubport**→**advertisedValue()****hubport.get_advertisedValue()**

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

hubport→**get_baudRate()**

YHubPort

hubport→**baudRate()****hubport.get_baudRate()**

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

```
function get_baudRate( )
```

La valeur par défaut est 1000 kbps, une valeur inférieure révèle des problèmes de communication.

Retourne :

un entier représentant la vitesse de transfert utilisée par le port de Yocto-hub, en kbps

En cas d'erreur, déclenche une exception ou retourne Y_BAUDRATE_INVALID.

hubport→**get_enabled()****YHubPort****hubport**→**enabled()****hubport.get_enabled()**

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

```
function get_enabled( )
```

Retourne :

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon vrai si le port du Yocto-hub est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

hubport→**get_errorMessage()**

YHubPort

hubport→**errorMessage()****hubport.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

hubport→**get_errorType()****YHubPort****hubport**→**errorType()****hubport.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

hubport→**get_friendlyName()**

YHubPort

hubport→**friendlyName()****hubport.get_friendlyName()**

Retourne un identifiant global du port de Yocto-hub au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du port de Yocto-hub si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port de Yocto-hub (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le port de Yocto-hub en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

hubport→**get_functionDescriptor()****YHubPort****hubport**→**functionDescriptor()****hubport.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

hubport→**get_functionId()**

YHubPort

hubport→**functionId()****hubport.get_functionId()**

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le port de Yocto-hub (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

hubport→**get_hardwareId()****YHubPort****hubport**→**hardwareId()****hubport.get_hardwareId()**

Retourne l'identifiant matériel unique du port de Yocto-hub au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port de Yocto-hub (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le port de Yocto-hub (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

hubport→**get_logicalName()**

YHubPort

hubport→**logicalName()****hubport.get_logicalName()**

Retourne le nom logique du port de Yocto-hub.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du port de Yocto-hub. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

hubport→**get_module()****YHubPort****hubport**→**module()****hubport.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

hubport→**get_module_async()**

YHubPort

hubport→**module_async()**

hubport.get_module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

hubport→**get_portState()****YHubPort****hubport**→**portState()****hubport.get_portState()**

Retourne l'état actuel du port de Yocto-hub.

```
function get_portState( )
```

Retourne :

une valeur parmi `Y_PORTSTATE_OFF`, `Y_PORTSTATE_OVRLD`, `Y_PORTSTATE_ON`, `Y_PORTSTATE_RUN` et `Y_PORTSTATE_PROG` représentant l'état actuel du port de Yocto-hub

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSTATE_INVALID`.

hubport→**get_userData()**

YHubPort

hubport→**userData()****hubport.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es 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( )
```

Si les valeurs des attributs en cache du port de Yocto-hub sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le port de Yocto-hub est joignable, `false` sinon

hubport→isOnline_async()hubport.isOnline_async()

YHubPort

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du port de Yocto-hub sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

hubport→**load()****hubport.load()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→**load_async()****hubport.load_async()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

hubport→**nextHubPort()****hubport.nextHubPort()****YHubPort**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

```
function nextHubPort( )
```

Retourne :

un pointeur sur un objet `YHubPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

hubport→registerValueCallback() hubport.registerValueCallback()

YHubPort

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

hubport→**set_enabled()****YHubPort****hubport**→**setEnabled()****hubport.set_enabled()**

Modifie le mode d'activation du port du Yocto-hub.

```
function set_enabled( newval)
```

Si le port est actif, il sera alimenté. Sinon, l'alimentation du module est coupée.

Paramètres :

newval soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon le mode d'activation du port du Yocto-hub

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→**set_logicalName()**

YHubPort

hubport→**setLogicalName()**

hubport.set_logicalName()

Modifie le nom logique du port de Yocto-hub.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du port de Yocto-hub.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→**set_userdata()****YHubPort****hubport**→**setUserData()****hubport.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

hubport→wait_async()hubport.wait_async()

YHubPort

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.20. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_humidity.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YHumidity = yoctolib.YHumidity;
php	require_once('yocto_humidity.php');
c++	#include "yocto_humidity.h"
m	#import "yocto_humidity.h"
pas	uses yocto_humidity;
vb	yocto_humidity.vb
cs	yocto_humidity.cs
java	import com.yoctopuce.YoctoAPI.YHumidity;
py	from yocto_humidity import *

Fonction globales

yFindHumidity(func)

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

Méthodes des objets YHumidity

humidity→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

humidity→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

humidity→get_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

humidity→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

humidity→get_currentValue()

Retourne la mesure actuelle de l'humidité.

humidity→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

humidity→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

humidity→get_friendlyName()

Retourne un identifiant global du capteur d'humidité au format `NOM_MODULE . NOM_FONCTION`.

humidity→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

humidity→get_functionId()

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

humidity→get_hardwareId()

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.

humidity→get_highestValue()

Retourne la valeur maximale observée pour l'humidité.

humidity→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.

humidity→get_logicalName()

Retourne le nom logique du capteur d'humidité.

humidity→get_lowestValue()

Retourne la valeur minimale observée pour l'humidité.

humidity→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

humidity→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

humidity→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.

humidity→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

humidity→get_resolution()

Retourne la résolution des valeurs mesurées.

humidity→get_unit()

Retourne l'unité dans laquelle l'humidité est exprimée.

humidity→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

humidity→isOnline()

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

humidity→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

humidity→load(msValidity)

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

humidity→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

humidity→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

humidity→nextHumidity()

Continue l'énumération des capteurs d'humidité commencée à l'aide de yFirstHumidity().

humidity→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

humidity→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

humidity→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée pour l'humidité.

humidity→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

humidity→set_logicalName(newval)

Modifie le nom logique du capteur d'humidité.

humidity→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée pour l'humidité.

humidity→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

humidity→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

humidity→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

humidity→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YHumidity.FindHumidity() yFindHumidity()yFindHumidity()

YHumidity

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

```
function yFindHumidity( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur d'humidité soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnline()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

Retourne :

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

**YHumidity.FirstHumidity()
yFirstHumidity()yFirstHumidity()**

YHumidity

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

```
function yFirstHumidity( )
```

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

Retourne :

un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

humidity→**calibrateFromPoints()**
humidity.calibrateFromPoints()**YHumidity**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→describe()humidity.describe()**YHumidity**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format `TYPE (NAME) =SERIAL .FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur d'humidité (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

humidity→**get_advertisedValue()**

YHumidity

humidity→**advertisedValue()**

humidity.get_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur d'humidité (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

humidity→**get_currentRawValue()****YHumidity****humidity**→**currentRawValue()****humidity.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

humidity→**get_currentValue()**

YHumidity

humidity→**currentValue()****humidity.get_currentValue()**

Retourne la mesure actuelle de l'humidité.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la mesure actuelle de l'humidité

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

humidity→**get_errorMessage()****YHumidity****humidity**→**errorMessage()****humidity.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

humidity→**get_errorType()**

YHumidity

humidity→**errorType()****humidity.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

humidity→get_friendlyName()**YHumidity****humidity→friendlyName()****humidity.get_friendlyName()**

Retourne un identifiant global du capteur d'humidité au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur d'humidité si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur d'humidité (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur d'humidité en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

humidity→**get_functionDescriptor()**
humidity→**functionDescriptor()**
humidity.get_functionDescriptor()

YHumidity

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

humidity→**get_functionId()****YHumidity****humidity**→**functionId()****humidity.get_functionId()**

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur d'humidité (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

humidity→get_hardwareId()

YHumidity

humidity→hardwareId()humidity.get_hardwareId()

Retourne l'identifiant matériel unique du capteur d'humidité au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur d'humidité (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur d'humidité (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

humidity→**get_highestValue()****YHumidity****humidity**→**highestValue()****humidity.get_highestValue()**

Retourne la valeur maximale observée pour l'humidité.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour l'humidité

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

humidity→get_logFrequency()

YHumidity

humidity→logFrequency()

humidity.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

humidity→**get_logicalName()****YHumidity****humidity**→**logicalName()****humidity.get_logicalName()**

Retourne le nom logique du capteur d'humidité.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur d'humidité. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

humidity→**get_lowestValue()**

YHumidity

humidity→**lowestValue()****humidity.get_lowestValue()**

Retourne la valeur minimale observée pour l'humidité.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour l'humidité

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

humidity→**get_module()****YHumidity****humidity**→**module()****humidity.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

humidity→**get_module_async()**

YHumidity

humidity→**module_async()**

humidity.get_module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

humidity→**get_recordedData()****YHumidity****humidity**→**recordedData()****humidity.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

humidity→**get_reportFrequency()**

YHumidity

humidity→**reportFrequency()**

humidity.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

humidity→**get_resolution()****YHumidity****humidity**→**resolution()****humidity.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

humidity→**get_unit()**

YHumidity

humidity→**unit()****humidity.get_unit()**

Retourne l'unité dans laquelle l'humidité est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle l'humidité est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

humidity→**get_userdata()****YHumidity****humidity**→**userData()****humidity.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

humidity→isOnline()humidity.isOnline()

YHumidity

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur d'humidité sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur d'humidité est joignable, `false` sinon

humidity→isOnline_async()
humidity.isOnline_async()

YHumidity

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur d'humidité sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

humidity→load()humidity.load()

YHumidity

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→loadCalibrationPoints()
humidity.loadCalibrationPoints()**YHumidity**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→load_async(humidity.load_async())

YHumidity

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

humidity→**nextHumidity()****humidity.nextHumidity()****YHumidity**

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

```
function nextHumidity( )
```

Retourne :

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

humidity→registerTimedReportCallback() humidity.registerTimedReportCallback()

YHumidity

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

humidity→registerValueCallback()
humidity.registerValueCallback()

YHumidity

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

humidity→**set_highestValue()**

YHumidity

humidity→**setHighestValue()**

humidity.set_highestValue()

Modifie la mémoire de valeur maximale observée pour l'humidité.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour l'humidité

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_logFrequency()**YHumidity****humidity→setLogFrequency()****humidity.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→**set_logicalName()**
humidity→**setLogicalName()**
humidity.set_logicalName()

YHumidity

Modifie le nom logique du capteur d'humidité.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur d'humidité.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_lowestValue()
humidity→setLowestValue()
humidity.set_lowestValue()

YHumidity

Modifie la mémoire de valeur minimale observée pour l'humidité.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour l'humidité

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→**set_reportFrequency()**

YHumidity

humidity→**setReportFrequency()**

humidity.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→**set_resolution()****YHumidity****humidity**→**setResolution()****humidity.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→**set_userdata()**

YHumidity

humidity→**setUserData()****humidity.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

humidity→**wait_async()****humidity.wait_async()****YHumidity**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.21. Interface de la fonction Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la led à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_led.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YLed = yoctolib.YLed;</code>
php	<code>require_once('yocto_led.php');</code>
cpp	<code>#include "yocto_led.h"</code>
m	<code>#import "yocto_led.h"</code>
pas	<code>uses yocto_led;</code>
vb	<code>yocto_led.vb</code>
cs	<code>yocto_led.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YLed;</code>
py	<code>from yocto_led import *</code>

Fonction globales

yFindLed(func)

Permet de retrouver une led d'après un identifiant donné.

yFirstLed()

Commence l'énumération des leds accessibles par la librairie.

Méthodes des objets YLed

led→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

led→get_advertisedValue()

Retourne la valeur courante de la led (pas plus de 6 caractères).

led→get_blinking()

Retourne le mode de signalisation de la led.

led→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

led→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

led→get_friendlyName()

Retourne un identifiant global de la led au format `NOM_MODULE . NOM_FONCTION`.

led→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

led→get_functionId()

Retourne l'identifiant matériel de la led, sans référence au module.

led→get_hardwareId()

Retourne l'identifiant matériel unique de la led au format `SERIAL . FUNCTIONID`.

led→get_logicalName()

Retourne le nom logique de la led.

led→get_luminosity()

Retourne l'intensité de la led en pour cent.

led→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

led→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

led→get_power()

Retourne l'état courant de la led.

led→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

led→isOnline()

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

led→isOnline_async(callback, context)

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

led→load(msValidity)

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

led→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

led→nextLed()

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

led→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

led→set_blinking(newval)

Modifie le mode de signalisation de la led.

led→set_logicalName(newval)

Modifie le nom logique de la led.

led→set_luminosity(newval)

Modifie l'intensité lumineuse de la led (en pour cent).

led→set_power(newval)

Modifie l'état courant de la led.

led→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

led→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YLed.FindLed() yFindLed()yFindLed()

YLed

Permet de retrouver une led d'après un identifiant donné.

```
function yFindLed( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la led soit en ligne au moment ou 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( )
```

Utiliser la fonction `YLed.nextLed()` pour itérer sur les autres leds.

Retourne :

un pointeur sur un objet `YLed`, correspondant à la première led accessible en ligne, ou `null` si il n'y a pas de leds disponibles.

led→describe()led.describe()**YLed**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE (NAME) =SERIAL .FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant la led (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

led→**get_advertisedValue()****YLed****led**→**advertisedValue()****led.get_advertisedValue()**

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

led→**get_blinking()**

YLed

led→**blinking()****led.get_blinking()**

Retourne le mode de signalisation de la led.

```
function get_blinking( )
```

Retourne :

une valeur parmi `Y_BLINKING_STILL`, `Y_BLINKING_RELAX`, `Y_BLINKING_AWARE`, `Y_BLINKING_RUN`, `Y_BLINKING_CALL` et `Y_BLINKING_PANIC` représentant le mode de signalisation de la led

En cas d'erreur, déclenche une exception ou retourne `Y_BLINKING_INVALID`.

led→`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( )
```

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( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la led.

led→**get_friendlyName()****YLed****led**→**friendlyName()****led.get_friendlyName()**

Retourne un identifiant global de la led au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la led si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la led (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la led en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

led→**get_functionDescriptor()**

YLed

led→**functionDescriptor()**

led.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

led→`get_functionId()`

YLed

led→`functionId()`**led**.`get_functionId()`

Retourne l'identifiant matériel de la led, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la led (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

led→**get_hardwareId()**

YLed

led→**hardwareId()****led.get_hardwareId()**

Retourne l'identifiant matériel unique de la led au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant la led (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

led→**get_logicalName()****YLed****led**→**logicalName()****led.get_logicalName()**

Retourne le nom logique de la led.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de la led. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

led→**get_luminosity()**

YLed

led→**luminosity()****led.get_luminosity()**

Retourne l'intensité de la led en pour cent.

```
function get_luminosity( )
```

Retourne :

un entier représentant l'intensité de la led en pour cent

En cas d'erreur, déclenche une exception ou retourne `Y_LUMINOSITY_INVALID`.

led→`get_module()`**YLed****led**→`module()`**led**.`get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

led→**get_module_async()**

YLed

led→**module_async()****led.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

led→**get_power()****YLed****led**→**power()****led.get_power()**

Retourne l'état courant de la led.

```
function get_power( )
```

Retourne :

soit `Y_POWER_OFF`, soit `Y_POWER_ON`, selon l'état courant de la led

En cas d'erreur, déclenche une exception ou retourne `Y_POWER_INVALID`.

led→**get_userData()**

YLed

led→**userData()****led.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

led→**isOnline()**(**led.isOnline()**)**YLed**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de la led sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si la led est joignable, `false` sinon

led→isOnline_async()led.isOnline_async()

YLed

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de la led sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

led→**load()****led.load()****YLed**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**load_async()****led.load_async()****YLed**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

led→**nextLed()****led.nextLed()****YLed**

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

```
function nextLed( )
```

Retourne :

un pointeur sur un objet `YLed` accessible en ligne, ou `null` lorsque l'énumération est terminée.

led→**registerValueCallback()**
led.registerValueCallback()**YLed**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

led→**set_blinking()****YLed****led**→**setBlinking()****led.set_blinking()**

Modifie le mode de signalisation de la led.

```
function set_blinking( newval)
```

Paramètres :

newval une valeur parmi Y_BLINKING_STILL, Y_BLINKING_RELAX, Y_BLINKING_AWARE, Y_BLINKING_RUN, Y_BLINKING_CALL et Y_BLINKING_PANIC représentant le mode de signalisation de la led

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_logicalName()**

YLed

led→**setLogicalName()****led.set_logicalName()**

Modifie le nom logique de la led.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la led.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_luminosity()****YLed****led**→**setLuminosity()****led.set_luminosity()**

Modifie l'intensité lumineuse de la led (en pour cent).

```
function set_luminosity( newval)
```

Paramètres :

newval un entier représentant l'intensité lumineuse de la led (en pour cent)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_power()**

YLed

led→**setPower()****led.set_power()**

Modifie l'état courant de la led.

```
function set_power( newval)
```

Paramètres :

newval soit Y_POWER_OFF, soit Y_POWER_ON, selon l'état courant de la led

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_userdata()****YLed****led**→**setUserData()****led.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

led→wait_async()led.wait_async()

YLed

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.22. Interface de la fonction LightSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_lightsensor.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YLightSensor = yoctolib.YLightSensor;
php	require_once('yocto_lightsensor.php');
cpp	#include "yocto_lightsensor.h"
m	#import "yocto_lightsensor.h"
pas	uses yocto_lightsensor;
vb	yocto_lightsensor.vb
cs	yocto_lightsensor.cs
java	import com.yoctopuce.YoctoAPI.YLightSensor;
py	from yocto_lightsensor import *

Fonction globales

yFindLightSensor(func)

Permet de retrouver un capteur de lumière d'après un identifiant donné.

yFirstLightSensor()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

Méthodes des objets YLightSensor

lightsensor→calibrate(calibratedVal)

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

lightsensor→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

lightsensor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format TYPE (NAME) =SERIAL . FUNCTIONID.

lightsensor→get_advertisedValue()

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

lightsensor→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

lightsensor→get_currentValue()

Retourne la mesure actuelle de la lumière ambiante.

lightsensor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

lightsensor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

lightsensor→get_friendlyName()

Retourne un identifiant global du capteur de lumière au format NOM_MODULE . NOM_FONCTION.

lightsensor→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

lightsensor→get_functionId()

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

lightsensor→**get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de lumière au format `SERIAL.FUNCTIONID`.

lightsensor→**get_highestValue()**

Retourne la valeur maximale observée pour la lumière ambiante.

lightsensor→**get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

lightsensor→**get_logicalName()**

Retourne le nom logique du capteur de lumière.

lightsensor→**get_lowestValue()**

Retourne la valeur minimale observée pour la lumière ambiante.

lightsensor→**get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

lightsensor→**get_module_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

lightsensor→**get_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.

lightsensor→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

lightsensor→**get_resolution()**

Retourne la résolution des valeurs mesurées.

lightsensor→**get_unit()**

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

lightsensor→**get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

lightsensor→**isOnline()**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

lightsensor→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

lightsensor→**load(msValidity)**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

lightsensor→**loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

lightsensor→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

lightsensor→**nextLightSensor()**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

lightsensor→**registerTimedReportCallback(callback)**

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 pour la lumière ambiante.

lightsensor→**set_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

lightsensor→**set_logicalName(newval)**

Modifie le nom logique du capteur de lumière.

lightsensor→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

lightsensor→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

lightsensor→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

lightsensor→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

lightsensor→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YLightSensor.FindLightSensor() yFindLightSensor()yFindLightSensor()

YLightSensor

Permet de retrouver un capteur de lumière d'après un identifiant donné.

```
function yFindLightSensor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de lumière soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLightSensor.isOnline()` pour tester si le capteur de lumière est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de lumière sans ambiguïté

Retourne :

un objet de classe `YLightSensor` qui permet ensuite de contrôler le capteur de lumière.

**YLightSensor.FirstLightSensor()
yFirstLightSensor()yFirstLightSensor()**

YLightSensor

Commence l'énumération des capteurs de lumière accessibles par la librairie.

```
function yFirstLightSensor( )
```

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

Retourne :

un pointeur sur un objet `YLightSensor`, correspondant au premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

lightsensor→calibrate()lightsensor.calibrate()

YLightSensor

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

```
function calibrate( calibratedVal)
```

Paramètres :

calibratedVal la consigne de valeur désirée.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**calibrateFromPoints()**
lightsensor.calibrateFromPoints()**YLightSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→describe()lightsensor.describe()

YLightSensor

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de lumière (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

lightsensor→**get_advertisedValue()****YLightSensor****lightsensor**→**advertisedValue()****lightsensor.get_advertisedValue()**

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de lumière (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

lightsensor→get_currentRawValue()

YLightSensor

lightsensor→currentRawValue()

lightsensor.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

lightsensor→get_currentValue()**YLightSensor****lightsensor→currentValue()****lightsensor.get_currentValue()**

Retourne la mesure actuelle de la lumière ambiante.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la mesure actuelle de la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

lightsensor→get_errorMessage()

YLightSensor

lightsensor→errorMessage()

lightsensor.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

lightsensor→**get_errorType()****YLightSensor****lightsensor**→**errorType()****lightsensor.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

lightsensor→get_friendlyName()

YLightSensor

lightsensor→friendlyName()

lightsensor.get_friendlyName()

Retourne un identifiant global du capteur de lumière au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de lumière si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de lumière (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de lumière en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

lightsensor→**get_functionDescriptor()**
lightsensor→**functionDescriptor()**
lightsensor.get_functionDescriptor()

YLightSensor

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

lightsensor→**get_functionId()**

YLightSensor

lightsensor→**functionId()****lightsensor.get_functionId()**

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de lumière (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

lightsensor→**get_hardwareId()****YLightSensor****lightsensor**→**hardwareId()****lightsensor.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de lumière au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de lumière (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de lumière (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

lightsensor→get_highestValue()

YLightSensor

lightsensor→highestValue()

lightsensor.get_highestValue()

Retourne la valeur maximale observée pour la lumière ambiante.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

lightsensor→get_logFrequency()**YLightSensor****lightsensor→logFrequency()****lightsensor.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

lightsensor→**get_logicalName()**

YLightSensor

lightsensor→**logicalName()**

lightsensor.get_logicalName()

Retourne le nom logique du capteur de lumière.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de lumière. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

lightsensor→**get_lowestValue()****YLightSensor****lightsensor**→**lowestValue()****lightsensor.get_lowestValue()**

Retourne la valeur minimale observée pour la lumière ambiante.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

lightsensor→**get_module()**

YLightSensor

lightsensor→**module()****lightsensor.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

lightsensor→**get_module_async()****YLightSensor****lightsensor**→**module_async()****lightsensor.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

lightsensor→**get_recordedData()**

YLightSensor

lightsensor→**recordedData()**

lightsensor.get_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

lightsensor→**get_reportFrequency()****YLightSensor****lightsensor**→**reportFrequency()****lightsensor.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

lightsensor→**get_resolution()**

YLightSensor

lightsensor→**resolution()****lightsensor.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

lightsensor→**get_unit()****YLightSensor****lightsensor**→**unit()****lightsensor.get_unit()**

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la lumière ambiante est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

lightsensor→**get_userData()**

YLightSensor

lightsensor→**userData()****lightsensor.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

lightsensor→**isOnline()****lightsensor.isOnline()****YLightSensor**

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de lumière sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de lumière est joignable, `false` sinon

lightsensor→isOnline_async() lightsensor.isOnline_async()

YLightSensor

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de lumière sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

lightsensor→**load()****lightsensor.load()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→loadCalibrationPoints() lightsensor.loadCalibrationPoints()

YLightSensor

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**load_async()****lightsensor.load_async()****YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

**lightsensor→nextLightSensor()
lightsensor.nextLightSensor()**

YLightSensor

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

```
function nextLightSensor( )
```

Retourne :

un pointeur sur un objet `YLightSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

lightsensor→registerTimedReportCallback()
lightsensor.registerTimedReportCallback()**YLightSensor**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

lightsensor→registerValueCallback() lightsensor.registerValueCallback()

YLightSensor

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

lightsensor→**set_highestValue()****YLightSensor****lightsensor**→**setHighestValue()****lightsensor.set_highestValue()**

Modifie la mémoire de valeur maximale observée pour la lumière ambiante.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour la lumière ambiante

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_logFrequency()

YLightSensor

lightsensor→setLogFrequency()

lightsensor.set_logFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**set_logicalName()****YLightSensor****lightsensor**→**setLogicalName()****lightsensor.set_logicalName()**

Modifie le nom logique du capteur de lumière.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de lumière.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_lowestValue()

YLightSensor

lightsensor→setLowestValue()

lightsensor.set_lowestValue()

Modifie la mémoire de valeur minimale observée pour la lumière ambiante.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour la lumière ambiante

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**set_reportFrequency()**
lightsensor→**setReportFrequency()**
lightsensor.set_reportFrequency()

YLightSensor

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**set_resolution()**

YLightSensor

lightsensor→**setResolution()**

lightsensor.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**set_userdata()****YLightSensor****lightsensor**→**setUserData()****lightsensor.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

lightsensor→wait_async()lightsensor.wait_async()

YLightSensor

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.23. Interface de la fonction Magnetometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_magnetometer.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YMagnetometer = yoctolib.YMagnetometer;
php	require_once('yocto_magnetometer.php');
cpp	#include "yocto_magnetometer.h"
m	#import "yocto_magnetometer.h"
pas	uses yocto_magnetometer;
vb	yocto_magnetometer.vb
cs	yocto_magnetometer.cs
java	import com.yoctopuce.YoctoAPI.YMagnetometer;
py	from yocto_magnetometer import *

Fonction globales

yFindMagnetometer(func)

Permet de retrouver un magnétomètre d'après un identifiant donné.

yFirstMagnetometer()

Commence l'énumération des magnétomètres accessibles par la librairie.

Méthodes des objets YMagnetometer

magnetometer→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

magnetometer→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

magnetometer→get_advertisedValue()

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

magnetometer→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

magnetometer→get_currentValue()

Retourne la valeur actuelle du champ magnétique.

magnetometer→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

magnetometer→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

magnetometer→get_friendlyName()

Retourne un identifiant global du magnétomètre au format `NOM_MODULE . NOM_FONCTION`.

magnetometer→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

magnetometer→get_functionId()

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

magnetometer→get_hardwareId()

Retourne l'identifiant matériel unique du magnétomètre au format `SERIAL . FUNCTIONID`.

magnetometer→**get_highestValue()**

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

magnetometer→**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.

magnetometer→**get_logicalName()**

Retourne le nom logique du magnétomètre.

magnetometer→**get_lowestValue()**

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

magnetometer→**get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

magnetometer→**get_module_async(callback, context)**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

magnetometer→**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.

magnetometer→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

magnetometer→**get_resolution()**

Retourne la résolution des valeurs mesurées.

magnetometer→**get_unit()**

Retourne l'unité dans laquelle le champ magnétique est exprimée.

magnetometer→**get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

magnetometer→**get_xValue()**

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

magnetometer→**get_yValue()**

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

magnetometer→**get_zValue()**

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

magnetometer→**isOnline()**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

magnetometer→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

magnetometer→**load(msValidity)**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

magnetometer→**loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

magnetometer→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

magnetometer→**nextMagnetometer()**

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

magnetometer→**registerTimedReportCallback(callback)**

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(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

magnetometer→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YMagnetometer.FindMagnetometer() yFindMagnetometer()yFindMagnetometer()

YMagnetometer

Permet de retrouver un magnétomètre d'après un identifiant donné.

```
function yFindMagnetometer( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le magnétomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMagnetometer.isOnline()` pour tester si le magnétomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le magnétomètre sans ambiguïté

Retourne :

un objet de classe `YMagnetometer` qui permet ensuite de contrôler le magnétomètre.

**YMagnetometer.FirstMagnetometer()
yFirstMagnetometer()yFirstMagnetometer()**

YMagnetometer

Commence l'énumération des magnétomètres accessibles par la librairie.

```
function yFirstMagnetometer( )
```

Utiliser la fonction `YMagnetometer.nextMagnetometer()` pour itérer sur les autres magnétomètres.

Retourne :

un pointeur sur un objet `YMagnetometer`, correspondant au premier magnétomètre accessible en ligne, ou `null` si il n'y a pas de magnétomètres disponibles.

magnetometer→calibrateFromPoints() magnetometer.calibrateFromPoints()

YMagnetometer

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**describe()****magnetometer.describe()****YMagnetometer**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le magnétomètre (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

magnetometer→get_advertisedValue()

YMagnetometer

magnetometer→advertisedValue()

magnetometer.get_advertisedValue()

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du magnétomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

magnetometer→**get_currentRawValue()****YMagnetometer****magnetometer**→**currentRawValue()****magnetometer.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

magnetometer→get_currentValue()

YMagnetometer

magnetometer→currentValue()

magnetometer.get_currentValue()

Retourne la valeur actuelle du champ magnétique.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle du champ magnétique

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

magnetometer→**get_errorMessage()****YMagnetometer****magnetometer**→**errorMessage()****magnetometer.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

magnetometer→get_errorType()

YMagnetometer

magnetometer→errorType()

magnetometer.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

magnetometer→**get_friendlyName()****YMagnetometer****magnetometer**→**friendlyName()****magnetometer.get_friendlyName()**

Retourne un identifiant global du magnétomètre au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du magnétomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du magnétomètre (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le magnétomètre en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

magnetometer→**get_functionDescriptor()**

YMagnetometer

magnetometer→**functionDescriptor()**

magnetometer.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

magnetometer→**get_functionId()****YMagnetometer****magnetometer**→**functionId()****magnetometer.get_functionId()**

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le magnétomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

magnetometer→**get_hardwareId()**

YMagnetometer

magnetometer→**hardwareId()**

magnetometer.get_hardwareId()

Retourne l'identifiant matériel unique du magnétomètre au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du magnétomètre (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le magnétomètre (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

magnetometer→**get_highestValue()****YMagnetometer****magnetometer**→**highestValue()****magnetometer.get_highestValue()**

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

magnetometer→get_logFrequency()

YMagnetometer

magnetometer→logFrequency()

magnetometer.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

magnetometer→**get_logicalName()****YMagnetometer****magnetometer**→**logicalName()****magnetometer.get_logicalName()**

Retourne le nom logique du magnétomètre.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du magnétomètre. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

magnetometer→**get_lowestValue()**

YMagnetometer

magnetometer→**lowestValue()**

magnetometer.get_lowestValue()

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

magnetometer→**get_module()**
magnetometer→**module()**
magnetometer.get_module()

YMagnetometer

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

magnetometer→**get_module_async()**

YMagnetometer

magnetometer→**module_async()**

magnetometer.get_module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

magnetometer→**get_recordedData()****YMagnetometer****magnetometer**→**recordedData()****magnetometer.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

magnetometer→**get_reportFrequency()**

YMagnetometer

magnetometer→**reportFrequency()**

magnetometer.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

magnetometer→**get_resolution()****YMagnetometer****magnetometer**→**resolution()****magnetometer.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

magnetometer→**get_unit()**

YMagnetometer

magnetometer→**unit()****magnetometer.get_unit()**

Retourne l'unité dans laquelle le champ magnétique est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le champ magnétique est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

magnetometer→**get_userData()****YMagnetometer****magnetometer**→**userData()****magnetometer.userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

magnetometer→get_xValue()

YMagnetometer

magnetometer→xValue()magnetometer.get_xValue()

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

```
function get_xValue( )
```

Retourne :

une valeur numérique représentant la composante X du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_XVALUE_INVALID.

magnetometer→**get_yValue()****YMagnetometer****magnetometer**→**yValue()****magnetometer.get_yValue()**

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

```
function get_yValue( )
```

Retourne :

une valeur numérique représentant la composante Y du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_YVALUE_INVALID`.

magnetometer→get_zValue()

YMagnetometer

magnetometer→zValue()magnetometer.get_zValue()

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

```
function get_zValue( )
```

Retourne :

une valeur numérique représentant la composante Z du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_ZVALUE_INVALID.

magnetometer→**isOnline()****magnetometer.isOnline()****YMagnetometer**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du magnétomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le magnétomètre est joignable, `false` sinon

**magnetometer→isOnline_async()
magnetometer.isOnline_async()**

YMagnetometer

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du magnétomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

magnetometer→**load()****magnetometer.load()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**loadCalibrationPoints()**
magnetometer.loadCalibrationPoints()

YMagnetometer

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**load_async()**
magnetometer.load_async()**YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

magnetometer→**nextMagnetometer()**
magnetometer.nextMagnetometer()

YMagnetometer

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

```
function nextMagnetometer( )
```

Retourne :

un pointeur sur un objet `YMagnetometer` accessible en ligne, ou `null` lorsque l'énumération est terminée.

magnetometer→**registerTimedReportCallback()**
magnetometer.registerTimedReportCallback()

YMagnetometer

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

magnetometer→**registerValueCallback()** **magnetometer.registerValueCallback()**

YMagnetometer

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

`magnetometer`→`set_highestValue()`
`magnetometer`→`setHighestValue()`
`magnetometer.set_highestValue()`

YMagnetometer

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_logFrequency()**

YMagnetometer

magnetometer→**setLogFrequency()**

magnetometer.set_logFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_logicalName()****YMagnetometer****magnetometer**→**setLogicalName()****magnetometer.set_logicalName()**

Modifie le nom logique du magnétomètre.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du magnétomètre.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_lowestValue()**

YMagnetometer

magnetometer→**setLowestValue()**

magnetometer.set_lowestValue()

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_reportFrequency()**
magnetometer→**setReportFrequency()**
magnetometer.set_reportFrequency()

YMagnetometer

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_resolution()**

YMagnetometer

magnetometer→**setResolution()**

magnetometer.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_userdata()**
magnetometer→**setUserData()**
magnetometer.set_userdata()

YMagnetometer

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

magnetometer→**wait_async()**
magnetometer.wait_async()

YMagnetometer

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.24. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

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()**

YMeasure

measure→**averageValue()**

measure.get_averageValue()

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

```
function get_averageValue( )
```

Retourne :

un nombre décimal correspondant à la valeur moyenne observée.

measure→**get_endTimeUTC()****YMeasure****measure**→**endTimeUTC()****measure.get_endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

```
function get_endTimeUTC( )
```

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

measure→**get_maxValue()**

YMeasure

measure→**maxValue()****measure.get_maxValue()**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

```
function get_maxValue( )
```

Retourne :

un nombre décimal correspondant à la plus grande valeur observée.

measure→**get_minValue()****YMeasure****measure**→**minValue()****measure.get_minValue()**

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

```
function get_minValue( )
```

Retourne :

un nombre décimal correspondant à la plus petite valeur observée.

measure→**get_startTimeUTC()**

YMeasure

measure→**startTimeUTC()**

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).

```
function get_startTimeUTC( )
```

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et le début de la mesure.

3.25. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;
php	require_once('yocto_api.php');
cpp	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
java	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *

Fonction globales

yFindModule(func)

Permet de retrouver un module d'après son numéro de série ou son nom logique.

yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

Méthodes des objets YModule

module→describe()

Retourne un court texte décrivant le module.

module→download(pathname)

Télécharge le fichier choisi du module et retourne son contenu.

module→functionCount()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

module→functionId(functionIndex)

Retourne l'identifiant matériel de la *n*ème fonction du module.

module→functionName(functionIndex)

Retourne le nom logique de la *n*ème fonction du module.

module→functionValue(functionIndex)

Retourne la valeur publiée par la *n*ème fonction du module.

module→get_beacon()

Retourne l'état de la balise de localisation.

module→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

module→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

module→get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

module→get_hardwareId()

Retourne l'identifiant unique du module.

module→get_icon2d()

	Retourne l'icône du module.
module → get_lastLogs()	Retourne une chaîne de caractères contenant les derniers logs du module.
module → get_logicalName()	Retourne le nom logique du module.
module → get_luminosity()	Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).
module → get_persistentSettings()	Retourne l'état courant des réglages persistents du module.
module → get_productId()	Retourne l'identifiant USB du module, préprogrammé en usine.
module → get_productName()	Retourne le nom commercial du module, préprogrammé en usine.
module → get_productRelease()	Retourne le numéro de version matériel du module, préprogrammé en usine.
module → get_rebootCountdown()	Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.
module → get_serialNumber()	Retourne le numéro de série du module, préprogrammé en usine.
module → get_upTime()	Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module.
module → get_usbBandwidth()	Retourne le nombre d'interface USB utilisé par le module.
module → get_usbCurrent()	Retourne le courant consommé par le module sur le bus USB, en milliampères.
module → get_userData()	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
module → isOnline()	Vérifie si le module est joignable, sans déclencher d'erreur.
module → isOnline_async(callback, context)	Vérifie si le module est joignable, sans déclencher d'erreur.
module → load(msValidity)	Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
module → load_async(msValidity, callback, context)	Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
module → nextModule()	Continue l'énumération des modules commencée à l'aide de <code>yFirstModule()</code> .
module → reboot(secBeforeReboot)	Agende un simple redémarrage du module dans un nombre donné de secondes.
module → registerLogCallback(callback)	todo
module → revertFromFlash()	Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.
module → saveToFlash()	

Sauve les réglages courants dans la mémoire non volatile du module.

module→**set_beacon(newval)**

Allume ou éteint la balise de localisation du module.

module→**set_logicalName(newval)**

Change le nom logique du module.

module→**set_luminosity(newval)**

Modifie la luminosité des leds informatives du module.

module→**set_usbBandwidth(newval)**

Modifie le nombre d'interface USB utilisé par le module.

module→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

module→**triggerFirmwareUpdate(secBeforeReboot)**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YModule.FindModule() yFindModule()yFindModule()

YModule

Permet de retrouver un module d'après son numéro de série ou son nom logique.

```
function yFindModule( func)
```

Cette fonction n'exige pas que le module soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YModule.isOnline()` pour tester si le module est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

**YModule.FirstModule()
yFirstModule()yFirstModule()**

YModule

Commence l'énumération des modules accessibles par la librairie.

```
function yFirstModule( )
```

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :

un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

module→**describe()**(**module.describe()**)

YModule

Retourne un court texte décrivant le module.

function **describe**()

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

Retourne :

une chaîne de caractères décrivant le module

module→**download()****module.download()****YModule**

Télécharge le fichier choisi du module et retourne son contenu.

```
function download( pathname)
```

Paramètres :

pathname nom complet du fichier

Retourne :

le contenu du fichier chargé

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

module→**functionCount()****module.functionCount()**

YModule

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

```
function functionCount( )
```

Retourne :

le nombre de fonctions sur le module

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**functionId()****module.functionId()****YModule**

Retourne l'identifiant matériel de la *n*ème fonction du module.

```
function functionId( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→**functionName()****module.functionName()**

YModule

Retourne le nom logique de la *nième* fonction du module.

```
function functionName( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au nom logique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→**functionValue()****module.functionValue()****YModule**

Retourne la valeur publiée par la *n*ème fonction du module.

```
function functionValue( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→**get_beacon()**

YModule

module→**beacon()****module.get_beacon()**

Retourne l'état de la balise de localisation.

```
function get_beacon( )
```

Retourne :

soit `Y_BEACON_OFF`, soit `Y_BEACON_ON`, selon l'état de la balise de localisation

En cas d'erreur, déclenche une exception ou retourne `Y_BEACON_INVALID`.

module→**get_errorMessage()****YModule****module**→**errorMessage()****module.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

module→**get_errorType()**

YModule

module→**errorType()****module.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

module→**get_firmwareRelease()****YModule****module**→**firmwareRelease()****module.get_firmwareRelease()**

Retourne la version du logiciel embarqué du module.

```
function get_firmwareRelease( )
```

Retourne :

une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne `Y_FIRMWARERELEASE_INVALID`.

module→**get_hardwareId()**

YModule

module→**hardwareId()****module.get_hardwareId()**

Retourne l'identifiant unique du module.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

Retourne :

une chaîne de caractères identifiant la fonction

module→**get_icon2d()****YModule****module**→**icon2d()****module.get_icon2d()**

Retourne l'icône du module.

```
function get_icon2d( )
```

L'icone est au format PNG et a une taille maximale de 1536 octets.

Retourne :

un buffer binaire contenant l'icone, au format png.

module→**get_lastLogs()**

YModule

module→**lastLogs()****module.get_lastLogs()**

Retourne une chaîne de caractère contenant les derniers logs du module.

```
function get_lastLogs( )
```

Cette méthode retourne les derniers logs qui sont encore stockés dans le module.

Retourne :

une chaîne de caractère contenant les derniers logs du module.

module→**get_logicalName()****YModule****module**→**logicalName()****module.get_logicalName()**

Retourne le nom logique du module.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

module→**get_luminosity()**

YModule

module→**luminosity()****module.get_luminosity()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
function get_luminosity( )
```

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne `Y_LUMINOSITY_INVALID`.

module→**get_persistentSettings()****YModule****module**→**persistentSettings()****module.get_persistentSettings()**

Retourne l'état courant des réglages persistents du module.

```
function get_persistentSettings( )
```

Retourne :

une valeur parmi `Y_PERSISTENTSETTINGS_LOADED`, `Y_PERSISTENTSETTINGS_SAVED` et `Y_PERSISTENTSETTINGS_MODIFIED` représentant l'état courant des réglages persistents du module

En cas d'erreur, déclenche une exception ou retourne `Y_PERSISTENTSETTINGS_INVALID`.

module→**get_productId()**

YModule

module→**productId()****module.get_productId()**

Retourne l'identifiant USB du module, préprogrammé en usine.

```
function get_productId( )
```

Retourne :

un entier représentant l'identifiant USB du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_PRODUCTID_INVALID`.

module→**get_productName()****YModule****module**→**productName()****module.get_productName()**

Retourne le nom commercial du module, préprogrammé en usine.

```
function get_productName( )
```

Retourne :

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_PRODUCTNAME_INVALID`.

module→**get_productRelease()**

YModule

module→**productRelease()**

module.get_productRelease()

Retourne le numéro de version matériel du module, préprogrammé en usine.

```
function get_productRelease( )
```

Retourne :

un entier représentant le numéro de version matériel du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_PRODUCTRELEASE_INVALID`.

module→**get_rebootCountdown()****YModule****module**→**rebootCountdown()****module.get_rebootCountdown()**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

```
function get_rebootCountdown( )
```

Retourne :

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

En cas d'erreur, déclenche une exception ou retourne `Y_REBOOTCOUNTDOWN_INVALID`.

module→**get_serialNumber()**

YModule

module→**serialNumber()****module.get_serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

```
function get_serialNumber( )
```

Retourne :

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_SERIALNUMBER_INVALID`.

module→**get_upTime()****YModule****module**→**upTime()****module.get_upTime()**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

```
function get_upTime( )
```

Retourne :

un entier représentant le nombre de millisecondes écoulées depuis la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_UPTIME_INVALID`.

module→**get_usbBandwidth()**

YModule

module→**usbBandwidth()**

module.get_usbBandwidth()

Retourne le nombre d'interface USB utilisé par le module.

```
function get_usbBandwidth( )
```

Retourne :

soit `Y_USBBANDWIDTH_SIMPLE`, soit `Y_USBBANDWIDTH_DOUBLE`, selon le nombre d'interface USB utilisé par le module

En cas d'erreur, déclenche une exception ou retourne `Y_USBBANDWIDTH_INVALID`.

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( )
```

Retourne :

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

En cas d'erreur, déclenche une exception ou retourne `Y_USBCURRENT_INVALID`.

module→**get_userData()**

YModule

module→**userData()****module.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

module→**isOnline()****module.isOnline()****YModule**

Vérifie si le module est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le module est joignable, `false` sinon

module→**isOnline_async()****module.isOnline_async()**

YModule

Vérifie si le module est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le résultat booléen
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

module→**load()****module.load()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**load_async()****module.load_async()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

module→**nextModule()****module.nextModule()****YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

```
function nextModule( )
```

Retourne :

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

module→**reboot()**(**module.reboot()**)

YModule

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
function reboot( secBeforeReboot)
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**revertFromFlash()**
module.revertFromFlash()

YModule

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

```
function revertFromFlash( )
```

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**saveToFlash()****module.saveToFlash()**

YModule

Sauve les réglages courants dans la mémoire non volatile du module.

```
function saveToFlash( )
```

Attention le nombre total de sauvegardes possibles durant la vie du module est limité (environ 100000 cycles). N'appellez pas cette fonction dans une boucle.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_beacon()****YModule****module**→**setBeacon()****module.set_beacon()**

Allume ou éteint la balise de localisation du module.

```
function set_beacon( newval)
```

Paramètres :

newval soit Y_BEACON_OFF, soit Y_BEACON_ON

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_logicalName()**

YModule

module→**setLogicalName()****module.set_logicalName()**

Change le nom logique du module.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_luminosity()****YModule****module**→**setLuminosity()****module.set_luminosity()**

Modifie la luminosité des leds informatives du module.

```
function set_luminosity( newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité des leds informatives du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_usbBandwidth()**

YModule

module→**setUsbBandwidth()**

module.set_usbBandwidth()

Modifie le nombre d'interface USB utilisé par le module.

```
function set_usbBandwidth( newval)
```

Vous devez redémarrer le module après avoir changé ce réglage.

Paramètres :

newval soit Y_USBBANDWIDTH_SIMPLE, soit Y_USBBANDWIDTH_DOUBLE, selon le nombre d'interface USB utilisé par le module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_userdata()****YModule****module**→**setUserData()****module.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

module→**triggerFirmwareUpdate()**
module.triggerFirmwareUpdate()

YModule

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

```
function triggerFirmwareUpdate( secBeforeReboot)
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**wait_async()****module.wait_async()****YModule**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.26. Interface de la fonction Network

Les objets YNetwork permettent de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_network.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YNetwork = yoctolib.YNetwork;
php	require_once('yocto_network.php');
c++	#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 laisser-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 laisser-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 connexion WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

network→isOnline()

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

network→isOnline_async(callback, context)

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

network→load(msValidity)

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

network→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

network→nextNetwork()

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

network→ping(host)

Ping `str_host` pour vérifier la connexion réseau.

network→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

network→set_adminPassword(newval)

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

network→set_callbackCredentials(newval)

Modifie le laisser-passer pour se connecter à l'adresse de callback.

network→set_callbackEncoding(newval)

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

network→set_callbackMaxDelay(newval)

Modifie l'attente maximale entre deux notifications par callback, en secondes.

network→set_callbackMethod(newval)

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

network→set_callbackMinDelay(newval)

Modifie l'attente minimale entre deux notifications par callback, en secondes.

network→set_callbackUrl(newval)

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

network→set_discoverable(newval)

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

network→set_logicalName(newval)

Modifie le nom logique de l'interface réseau.

network→set_primaryDNS(newval)

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→set_secondaryDNS(newval)

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

network→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

network→set_userPassword(newval)

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

network→set_wwwWatchdogDelay(newval)

Modifie la durée de perte de connexion WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

network→**useStaticIP(ipAddress, subnetMaskLen, router)**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

network→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YNetwork.FindNetwork() yFindNetwork()yFindNetwork()

YNetwork

Permet de retrouver une interface réseau d'après un identifiant donné.

```
function yFindNetwork( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YNetwork.isOnline()` pour tester si l'interface réseau est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'interface réseau sans ambiguïté

Retourne :

un objet de classe `YNetwork` qui permet ensuite de contrôler l'interface réseau.

**YNetwork.FirstNetwork()
yFirstNetwork()yFirstNetwork()**

YNetwork

Commence l'énumération des interfaces réseau accessibles par la librairie.

```
function yFirstNetwork( )
```

Utiliser la fonction `YNetwork.nextNetwork()` pour itérer sur les autres interfaces réseau.

Retourne :

un pointeur sur un objet `YNetwork`, correspondant à la première interface réseau accessible en ligne, ou `null` si il n'y a pas de interfaces réseau disponibles.

network→callbackLogin()**network.callbackLogin()**

YNetwork

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

```
function callbackLogin( username, password)
```

Le mot de passe ne sera pas stocké dans le module, mais seulement une version hashée non réversible. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

username nom d'utilisateur pour s'identifier au callback

password mot de passe pour s'identifier au callback

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→describe()network.describe()**YNetwork**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.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 debugueur.

Retourne :

une chaîne de caractères décrivant l'interface réseau (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

network→**get_adminPassword()**

YNetwork

network→**adminPassword()**

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.

```
function get_adminPassword( )
```

Retourne :

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_ADMINPASSWORD_INVALID`.

network→**get_advertisedValue()****YNetwork****network**→**advertisedValue()****network.get_advertisedValue()**

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'interface réseau (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

network→**get_callbackCredentials()**

YNetwork

network→**callbackCredentials()**

network.get_callbackCredentials()

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

```
function get_callbackCredentials( )
```

Retourne :

une chaîne de caractères représentant une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKCREDENTIALS_INVALID`.

network→**get_callbackEncoding()****YNetwork****network**→**callbackEncoding()****network.get_callbackEncoding()**

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
function get_callbackEncoding( )
```

Retourne :

une valeur parmi `Y_CALLBACKENCODING_FORM`, `Y_CALLBACKENCODING_JSON`, `Y_CALLBACKENCODING_JSON_ARRAY`, `Y_CALLBACKENCODING_CSV` et `Y_CALLBACKENCODING_YOCTO_API` représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKENCODING_INVALID`.

network→**get_callbackMaxDelay()**

YNetwork

network→**callbackMaxDelay()**

network.get_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

```
function get_callbackMaxDelay( )
```

Retourne :

un entier représentant l'attente maximale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMAXDELAY_INVALID`.

network→**get_callbackMethod()****YNetwork****network**→**callbackMethod()****network.get_callbackMethod()**

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
function get_callbackMethod( )
```

Retourne :

une valeur parmi `Y_CALLBACKMETHOD_POST`, `Y_CALLBACKMETHOD_GET` et `Y_CALLBACKMETHOD_PUT` représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMETHOD_INVALID`.

network→get_callbackMinDelay()

YNetwork

network→callbackMinDelay()

network.get_callbackMinDelay()

Retourne l'attente minimale entre deux notifications par callback, en secondes.

```
function get_callbackMinDelay( )
```

Retourne :

un entier représentant l'attente minimale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMINDELAY_INVALID`.

network→**get_callbackUrl()****YNetwork****network**→**callbackUrl()****network.get_callbackUrl()**

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
function get_callbackUrl( )
```

Retourne :

une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKURL_INVALID`.

network→**get_discoverable()**

YNetwork

network→**discoverable()****network.get_discoverable()**

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

```
function get_discoverable( )
```

Retourne :

soit `Y_DISCOVERABLE_FALSE`, soit `Y_DISCOVERABLE_TRUE`, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne `Y_DISCOVERABLE_INVALID`.

network→**get_errorMessage()****YNetwork****network**→**errorMessage()****network.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

network→**get_errorType()**

YNetwork

network→**errorType()****network.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

network→**get_friendlyName()****YNetwork****network**→**friendlyName()****network.get_friendlyName()**

Retourne un identifiant global de l'interface réseau au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'interface réseau si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'interface réseau en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

network→**get_functionDescriptor()**
network→**functionDescriptor()**
network.get_functionDescriptor()

YNetwork

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

network→**get_functionId()****YNetwork****network**→**functionId()****network.get_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'interface réseau (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

network→**get_hardwareId()**

YNetwork

network→**hardwareId()****network.get_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'interface réseau (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

network→**get_ipAddress()****YNetwork****network**→**ipAddress()****network.get_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

```
function get_ipAddress( )
```

Il peut s'agir d'une adresse configurée statiquement, ou d'une adresse reçue par un serveur DHCP.

Retourne :

une chaîne de caractères représentant l'adresse IP utilisée par le module Yoctopuce

En cas d'erreur, déclenche une exception ou retourne `Y_IPADDRESS_INVALID`.

network→**get_logicalName()**

YNetwork

network→**logicalName()****network.get_logicalName()**

Retourne le nom logique de l'interface réseau.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'interface réseau. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

network→**get_macAddress()****YNetwork****network**→**macAddress()****network.get_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

```
function get_macAddress( )
```

L'adresse MAC est aussi présente sur un autocollant sur le module, représentée en chiffres et en code-barres.

Retourne :

une chaîne de caractères représentant l'adresse MAC de l'interface réseau, unique pour chaque module

En cas d'erreur, déclenche une exception ou retourne `Y_MACADDRESS_INVALID`.

network→**get_module()**

YNetwork

network→**module()****network.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

network→**get_module_async()****YNetwork****network**→**module_async()****network.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

network→**get_poeCurrent()**

YNetwork

network→**poeCurrent()****network.get_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

```
function get_poeCurrent( )
```

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

Retourne :

un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne Y_POECURRENT_INVALID.

network→**get_primaryDNS()****YNetwork****network**→**primaryDNS()****network.get_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
function get_primaryDNS( )
```

Retourne :

une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne `Y_PRIMARYDNS_INVALID`.

network→**get_readiness()****YNetwork****network**→**readiness()****network.get_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

```
function get_readiness( )
```

Le niveau zéro (DOWN_0) signifie qu'aucun support réseau matériel n'a été détecté. Soit il n'y a pas de signal sur le câble réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme l'existence du SSID configuré. Le niveau 2 (LINK_2) est atteint lorsque le support matériel du réseau est fonctionnel. Pour une connection réseau filaire, le niveau 2 signifie que le câble est connecté aux deux bouts. Pour une connection à un point d'accès réseau sans fil, il démontre que les paramètres de sécurité configurés sont corrects. Pour une connection sans fil en mode ad-hoc, cela signifie qu'il y a au moins un partenaire sur le réseau ad-hoc. Le niveau 3 (DHCP_3) est atteint lorsque qu'une adresse IP a été obtenue par DHCP. Le niveau 4 (DNS_4) est atteint lorsqu'un serveur DNS est joignable par le réseau. Le niveau 5 (WWW_5) est atteint lorsque la connectivité globale à internet est avérée par l'obtention de l'heure courante sur un serveur NTP.

Retourne :

une valeur parmi Y_READINESS_DOWN, Y_READINESS_EXISTS, Y_READINESS_LINKED, Y_READINESS_LAN_OK et Y_READINESS_WWW_OK représentant l'état de fonctionnement atteint par l'interface réseau

En cas d'erreur, déclenche une exception ou retourne Y_READINESS_INVALID.

network→**get_router()****YNetwork****network**→**router()****network.get_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

```
function get_router( )
```

Retourne :

une chaîne de caractères représentant l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*)

En cas d'erreur, déclenche une exception ou retourne `Y_ROUTER_INVALID`.

network→get_secondaryDNS()
network→secondaryDNS()
network.get_secondaryDNS()

YNetwork

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

```
function get_secondaryDNS( )
```

Retourne :

une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne `Y_SECONDARYDNS_INVALID`.

network→**get_subnetMask()****YNetwork****network**→**subnetMask()****network.get_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

```
function get_subnetMask( )
```

Retourne :

une chaîne de caractères représentant le masque de sous-réseau utilisé par le module

En cas d'erreur, déclenche une exception ou retourne `Y_SUBNETMASK_INVALID`.

network→**get_userData()**

YNetwork

network→**userData()****network.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es 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()****YNetwork****network**→**userPassword()****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.

```
function get_userPassword( )
```

Retourne :

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_USERPASSWORD_INVALID`.

network→**get_wwwWatchdogDelay()**

YNetwork

network→**wwwWatchdogDelay()**

network.get_wwwWatchdogDelay()

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

```
function get_wwwWatchdogDelay( )
```

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW.

Retourne :

un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet

En cas d'erreur, déclenche une exception ou retourne `Y_WWWWATCHDOGDELAY_INVALID`.

network→**isOnline()****network.isOnline()****YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'interface réseau sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'interface réseau est joignable, `false` sinon

network→**isOnline_async()****network.isOnline_async()**

YNetwork

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'interface réseau sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

network→**load()****network.load()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**load_async()****network.load_async()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

network→**nextNetwork()****network.nextNetwork()****YNetwork**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

```
function nextNetwork( )
```

Retourne :

un pointeur sur un objet `YNetwork` accessible en ligne, ou `null` lorsque l'énumération est terminée.

network→ping()network.ping()

YNetwork

Ping str_host pour vérifier la connexion réseau.

```
function ping( host)
```

Envoie quatre requêtes ICMP ECHO_RESPONSER à la cible str_host depuis le module. Cette méthode retourne une chaîne de caractères avec le résultat des 4 requêtes ICMP ECHO_RESPONSE.

Paramètres :

host le nom d'hôte ou l'adresse IP de la cible

Retourne :

une chaîne de caractères contenant le résultat du ping.

network→registerValueCallback()
network.registerValueCallback()

YNetwork

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

network→**set_adminPassword()**
network→**setAdminPassword()**
network.set_adminPassword()

YNetwork

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

```
function set_adminPassword( newval)
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackCredentials()**
network→**setCallbackCredentials()**
network.set_callbackCredentials()

YNetwork

Modifie le laisser-passer pour se connecter à l'adresse de callback.

```
function set_callbackCredentials( newval)
```

Le laisser-passer doit être fourni tel que retourné par la fonction `get_callbackCredentials`, sous la forme `username:hash`. La valeur du hash dépend de la méthode d'autorisation implémentée par le callback. Pour une autorisation de type Basic, le hash est le MD5 de la chaîne `username:password`. Pour une autorisation de type Digest, le hash est le MD5 de la chaîne `username:realm:password`. Pour une utilisation simplifiée, utilisez la fonction `callbackLogin`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le laisser-passer pour se connecter à l'adresse de callback

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackEncoding()**
network→**setCallbackEncoding()**
network.set_callbackEncoding()

YNetwork

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
function set_callbackEncoding( newval)
```

Paramètres :

newval une valeur parmi `Y_CALLBACKENCODING_FORM`, `Y_CALLBACKENCODING_JSON`, `Y_CALLBACKENCODING_JSON_ARRAY`, `Y_CALLBACKENCODING_CSV` et `Y_CALLBACKENCODING_YOCTO_API` représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackMaxDelay()****YNetwork****network**→**setCallbackMaxDelay()****network.set_callbackMaxDelay()**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

```
function set_callbackMaxDelay( newval)
```

Paramètres :

newval un entier représentant l'attente maximale entre deux notifications par callback, en secondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackMethod()**

YNetwork

network→**setCallbackMethod()**

network.set_callbackMethod()

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
function set_callbackMethod( newval)
```

Paramètres :

newval une valeur parmi `Y_CALLBACKMETHOD_POST`, `Y_CALLBACKMETHOD_GET` et `Y_CALLBACKMETHOD_PUT` représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackMinDelay()****YNetwork****network**→**setCallbackMinDelay()****network.set_callbackMinDelay()**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

```
function set_callbackMinDelay( newval)
```

Paramètres :

newval un entier représentant l'attente minimale entre deux notifications par callback, en secondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackUrl()**

YNetwork

network→**setCallbackUrl()****network.set_callbackUrl()**

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
function set_callbackUrl( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_discoverable()****YNetwork****network**→**setDiscoverable()****network.set_discoverable()**

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

```
function set_discoverable( newval)
```

Paramètres :

newval soit `Y_DISCOVERABLE_FALSE`, soit `Y_DISCOVERABLE_TRUE`, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles 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()****YNetwork****network**→**setLogicalName()****network.set_logicalName()**

Modifie le nom logique de l'interface réseau.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'interface réseau.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_primaryDNS()****YNetwork****network**→**setPrimaryDNS()****network.set_primaryDNS()**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
function set_primaryDNS( newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_secondaryDNS()****YNetwork****network**→**setSecondaryDNS()****network.set_secondaryDNS()**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
function set_secondaryDNS( newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_userdata()****YNetwork****network**→**setUserData()****network.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

network→**set_userPassword()**
network→**setUserPassword()**
network.set_userPassword()

YNetwork

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

```
function set_userPassword( newval)
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_wwwWatchdogDelay()****YNetwork****network**→**setWwwWatchdogDelay()****network.set_wwwWatchdogDelay()**

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

```
function set_wwwWatchdogDelay( newval)
```

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW. La plus petite durée non-nulle utilisable est 90 secondes.

Paramètres :

newval un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→useDHCP()network.useDHCP()**YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

```
function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
```

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

- | | |
|------------------------------|---|
| fallbackIpAddr | adresse IP à utiliser si aucun serveur DHCP ne répond |
| fallbackSubnetMaskLen | longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0. |
| fallbackRouter | adresse de la passerelle à utiliser si aucun serveur DHCP ne répond |

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→useStaticIP(network.useStaticIP())**YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

```
function useStaticIP( ipAddress, subnetMaskLen, router)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ipAddress adresse IP à utiliser par le module
subnetMaskLen longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.
router adresse IP de la passerelle à utiliser ("default gateway")

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→wait_async()network.wait_async()

YNetwork

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.27. contrôle d'OS

L'objet `OsControl` permet de contrôler le système d'exploitation sur lequel tourne un VirtualHub. `OsControl` n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activé au lancement du VirtualHub, avec l'option `-o`.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_oscontrol.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YOsControl = yoctolib.YOsControl;</code>
php	<code>require_once('yocto_oscontrol.php');</code>
c++	<code>#include "yocto_oscontrol.h"</code>
m	<code>#import "yocto_oscontrol.h"</code>
pas	<code>uses yocto_oscontrol;</code>
vb	<code>yocto_oscontrol.vb</code>
cs	<code>yocto_oscontrol.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YOsControl;</code>
py	<code>from yocto_oscontrol import *</code>

Fonction globales

`yFindOsControl(func)`

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

`yFirstOsControl()`

Commence l'énumération des contrôle d'OS accessibles par la librairie.

Méthodes des objets `YOsControl`

`oscontrol→describe()`

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

`oscontrol→get_advertisedValue()`

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

`oscontrol→get_errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

`oscontrol→get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

`oscontrol→get_friendlyName()`

Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE . NOM_FONCTION`.

`oscontrol→get_functionDescriptor()`

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

`oscontrol→get_functionId()`

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

`oscontrol→get_hardwareId()`

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL . FUNCTIONID`.

`oscontrol→get_logicalName()`

Retourne le nom logique du contrôle d'OS.

`oscontrol→get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`oscontrol→get_module_async(callback, context)`

3. Référence

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(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`.

oscontrol→**shutdown(secBeforeShutDown)**

Agende un arrêt de l'OS dans un nombre donné de secondes.

oscontrol→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YOsControl.FindOsControl() yFindOsControl()yFindOsControl()

YOsControl

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

```
function yFindOsControl( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le contrôle d'OS soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YOsControl.isOnline()` pour tester si le contrôle d'OS est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

Retourne :

un objet de classe `YOsControl` qui permet ensuite de contrôler le contrôle d'OS.

YOsControl.FirstOsControl() yFirstOsControl()yFirstOsControl()

YOsControl

Commence l'énumération des contrôle d'OS accessibles par la librairie.

```
function yFirstOsControl( )
```

Utiliser la fonction `YOsControl.nextOsControl()` pour itérer sur les autres contrôle d'OS.

Retourne :

un pointeur sur un objet `YOsControl`, correspondant au premier contrôle d'OS accessible en ligne, ou `null` si il n'y a pas de contrôle d'OS disponibles.

oscontrol→describe(oscontrol.describe())**YOsControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le contrôle d'OS (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

oscontrol→**get_advertisedValue()**

YOsControl

oscontrol→**advertisedValue()**

oscontrol.get_advertisedValue()

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

oscontrol→get_errorMessage()**YOsControl****oscontrol→errorMessage()****oscontrol.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

oscontrol→**get_errorType()**

YOsControl

oscontrol→**errorType()****oscontrol.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

oscontrol→**get_friendlyName()****YOsControl****oscontrol**→**friendlyName()****oscontrol.get_friendlyName()**

Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du contrôle d'OS si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'OS (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le contrôle d'OS en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

oscontrol→**get_functionDescriptor()**
oscontrol→**functionDescriptor()**
oscontrol.get_functionDescriptor()

YOsControl

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

oscontrol→**get_functionId()****YOsControl****oscontrol**→**functionId()****oscontrol.get_functionId()**

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le contrôle d'OS (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

oscontrol→**get_hardwareId()**

YOsControl

oscontrol→**hardwareId()****oscontrol.get_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'OS (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le contrôle d'OS (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

oscontrol→**get_logicalName()****YOsControl****oscontrol**→**logicalName()****oscontrol.get_logicalName()**

Retourne le nom logique du contrôle d'OS.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du contrôle d'OS. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

oscontrol→**get_module()**

YOsControl

oscontrol→**module()****oscontrol.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

oscontrol→**get_module_async()**
oscontrol→**module_async()**
oscontrol.get_module_async()

YOsControl

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

oscontrol→get_shutdownCountdown()

YOsControl

oscontrol→shutdownCountdown()

oscontrol.get_shutdownCountdown()

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

```
function get_shutdownCountdown( )
```

Retourne :

un entier représentant le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé

En cas d'erreur, déclenche une exception ou retourne `Y_SHUTDOWNCOUNTDOWN_INVALID`.

oscontrol→**get_userdata()****YOsControl****oscontrol**→**userData()****oscontrol.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es 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.

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le contrôle d'OS est joignable, `false` sinon

oscontrol→**isOnline_async()**
oscontrol.isOnline_async()

YOsControl

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

oscontrol→**load()****oscontrol.load()****YOsControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→**load_async()****oscontrol.load_async()****YOsControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

oscontrol→**nextOsControl()**
oscontrol.nextOsControl()

YOsControl

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

```
function nextOsControl( )
```

Retourne :

un pointeur sur un objet `YOsControl` accessible en ligne, ou `null` lorsque l'énumération est terminée.

oscontrol→**registerValueCallback()**
oscontrol.registerValueCallback()

YOsControl

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

oscontrol→**set_logicalName()**
oscontrol→**setLogicalName()**
oscontrol.set_logicalName()

YOsControl

Modifie le nom logique du contrôle d'OS.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du contrôle d'OS.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→**set_userdata()****YOsControl****oscontrol**→**setUserData()****oscontrol.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

oscontrol→**shutdown()****oscontrol.shutdown()**

YOsControl

Agende un arrêt de l'OS dans un nombre donné de secondes.

```
function shutdown( secBeforeShutDown)
```

Paramètres :

secBeforeShutDown nombre de secondes avant l'arrêt

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→**wait_async()****oscontrol.wait_async()****YOsControl**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.28. Interface de la fonction Power

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_power.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YPower = yoctolib.YPower;</code>
php	<code>require_once('yocto_power.php');</code>
c++	<code>#include "yocto_power.h"</code>
m	<code>#import "yocto_power.h"</code>
pas	<code>uses yocto_power;</code>
vb	<code>yocto_power.vb</code>
cs	<code>yocto_power.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YPower;</code>
py	<code>from yocto_power import *</code>

Fonction globales

yFindPower(func)

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

yFirstPower()

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

Méthodes des objets YPower

power→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

power→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

power→get_advertisedValue()

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

power→get_cosPhi()

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

power→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

power→get_currentValue()

Retourne la valeur instantanée de la puissance électrique.

power→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

power→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

power→get_friendlyName()

Retourne un identifiant global du capteur de puissance électrique au format `NOM_MODULE . NOM_FONCTION`.

power→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

power→**get_functionId()**

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

power→**get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

power→**get_highestValue()**

Retourne la valeur maximale observée pour la puissance électrique.

power→**get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

power→**get_logicalName()**

Retourne le nom logique du capteur de puissance électrique.

power→**get_lowestValue()**

Retourne la valeur minimale observée pour la puissance électrique.

power→**get_meter()**

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

power→**get_meterTimer()**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

power→**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

power→**get_module_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

power→**get_recordedData(startTime, endTime)**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

power→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

power→**get_resolution()**

Retourne la résolution des valeurs mesurées.

power→**get_unit()**

Retourne l'unité dans laquelle la puissance électrique est exprimée.

power→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

power→**isOnline()**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

power→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

power→**load(msValidity)**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

power→**loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

power→**load_async(msValidity, callback, context)**

3. Reference

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

power→**nextPower()**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

power→**registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

power→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

power→**reset()**

Réinitialise le compteur d'énergie.

power→**set_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

power→**set_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

power→**set_logicalName(newval)**

Modifie le nom logique du capteur de puissance électrique.

power→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la puissance électrique.

power→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

power→**set_resolution(newval)**

Modifie la résolution des valeurs mesurées.

power→**set_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

power→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPower.FindPower() yFindPower()yFindPower()

YPower

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

```
function yFindPower( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de puissance électrique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance électrique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de puissance électrique sans ambiguïté

Retourne :

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance électrique.

YPower.FirstPower() yFirstPower(yFirstPower())

YPower

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

```
function yFirstPower( )
```

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance électrique.

Retourne :

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance électrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance électrique disponibles.

power→**calibrateFromPoints()** **power.calibrateFromPoints()**

YPower

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→describe()power.describe()

YPower

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de puissance électrique (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

power→**get_advertisedValue()****YPower****power**→**advertisedValue()****power.get_advertisedValue()**

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de puissance électrique (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

power→**get_cosPhi()**

YPower

power→**cosPhi()****power.get_cosPhi()**

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

```
function get_cosPhi( )
```

Retourne :

une valeur numérique représentant le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA)

En cas d'erreur, déclenche une exception ou retourne `Y_COSPHI_INVALID`.

power→**get_currentRawValue()****YPower****power**→**currentRawValue()****power.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

power→**get_currentValue()**

YPower

power→**currentValue()****power.get_currentValue()**

Retourne la valeur instantanée de la puissance électrique.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur instantanée de la puissance électrique

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

power→**get_errorMessage()****YPower****power**→**errorMessage()****power.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

power→**get_errorType()**

YPower

power→**errorType()****power.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

power→**get_friendlyName()****YPower****power**→**friendlyName()****power.get_friendlyName()**

Retourne un identifiant global du capteur de puissance électrique au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de puissance électrique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de puissance électrique (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de puissance électrique en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

power→**get_functionDescriptor()**

YPower

power→**functionDescriptor()**

power.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

power→**get_functionId()****YPower****power**→**functionId()****power.get_functionId()**

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

power→**get_hardwareId()**

YPower

power→**hardwareId()****power.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de puissance électrique (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de puissance électrique (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

power→**get_highestValue()****YPower****power**→**highestValue()****power.get_highestValue()**

Retourne la valeur maximale observée pour la puissance électrique.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

power→**get_logFrequency()**

YPower

power→**logFrequency()****power.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

power→**get_logicalName()****YPower****power**→**logicalName()****power.get_logicalName()**

Retourne le nom logique du capteur de puissance électrique.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de puissance électrique. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

power→**get_lowestValue()**

YPower

power→**lowestValue()****power.get_lowestValue()**

Retourne la valeur minimale observée pour la puissance électrique.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la puissance électrique

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

power→**get_meter()****YPower****power**→**meter()****power.get_meter()**

Retourne la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée.

```
function get_meter( )
```

Ce compteur est réinitialisé à chaque démarrage du module.

Retourne :

une valeur numérique représentant la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée

En cas d'erreur, déclenche une exception ou retourne `Y_METER_INVALID`.

power→**get_meterTimer()**

YPower

power→**meterTimer()****power.get_meterTimer()**

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

```
function get_meterTimer( )
```

Retourne :

un entier représentant le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_METERTIMER_INVALID`.

power→**get_module()****YPower****power**→**module()****power.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

power→**get_module_async()**

YPower

power→**module_async()****power.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

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( startTime, endTime)
```

Veillez 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()**

YPower

power→**reportFrequency()**

power.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

power→**get_resolution()****YPower****power**→**resolution()****power.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

power→**get_unit()**

YPower

power→**unit()****power.get_unit()**

Retourne l'unité dans laquelle la puissance électrique est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la puissance électrique est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

power→**get_userData()****YPower****power**→**userData()****power.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

power→isOnline()power.isOnline()

YPower

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de puissance électrique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de puissance électrique est joignable, `false` sinon

power→**isOnline_async()****power.isOnline_async()****YPower**

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context )
```

Si les valeurs des attributs en cache du capteur de puissance électrique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

power→**load()****power.load()****YPower**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→loadCalibrationPoints()
power.loadCalibrationPoints()

YPower

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→load_async()power.load_async()

YPower

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

power→**nextPower()****power.nextPower()****YPower**

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

```
function nextPower( )
```

Retourne :

un pointeur sur un objet `YPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

power→**registerTimedReportCallback()**
power.registerTimedReportCallback()

YPower

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

power→**registerValueCallback()**
power.registerValueCallback()

YPower

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

power→**reset()****power.reset()**

YPower

Réinitialise le compteur d'énergie.

function **reset**()

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_highestValue()****YPower****power**→**setHighestValue()****power.set_highestValue()**

Modifie la mémoire de valeur maximale observée pour la puissance électrique.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour la puissance électrique

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_logFrequency()**

YPower

power→**setLogFrequency()****power.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_logicalName()****YPower****power**→**setLogicalName()****power.set_logicalName()**

Modifie le nom logique du capteur de puissance électrique.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de puissance électrique.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_lowestValue()**

YPower

power→**setLowestValue()****power.set_lowestValue()**

Modifie la mémoire de valeur minimale observée pour la puissance électrique.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour la puissance électrique

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_reportFrequency()**
power→**setReportFrequency()**
power.set_reportFrequency()

YPower

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_resolution()**

YPower

power→**setResolution()****power.set_resolution()**

Modifie la résolution des valeurs mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_userdata()****YPower****power**→**setUserData()****power.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

power→wait_async()power.wait_async()

YPower

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.29. Interface de la fonction Pressure

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les 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');
c++	#include "yocto_pressure.h"
m	#import "yocto_pressure.h"
pas	uses yocto_pressure;
vb	yocto_pressure.vb
cs	yocto_pressure.cs
java	import com.yoctopuce.YoctoAPI.YPressure;
py	from yocto_pressure import *

Fonction globales

yFindPressure(func)

Permet de retrouver un capteur de pression d'après un identifiant donné.

yFirstPressure()

Commence l'énumération des capteurs de pression accessibles par la librairie.

Méthodes des objets YPressure

pressure→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

pressure→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

pressure→get_advertisedValue()

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

pressure→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

pressure→get_currentValue()

Retourne la mesure actuelle de la pression.

pressure→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

pressure→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

pressure→get_friendlyName()

Retourne un identifiant global du capteur de pression au format `NOM_MODULE . NOM_FONCTION`.

pressure→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

pressure→get_functionId()

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

pressure→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de pression au format SERIAL . FUNCTIONID.

pressure→**get_highestValue()**

Retourne la valeur maximale observée pour la pression.

pressure→**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.

pressure→**get_logicalName()**

Retourne le nom logique du capteur de pression.

pressure→**get_lowestValue()**

Retourne la valeur minimale observée pour la pression.

pressure→**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

pressure→**get_module_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

pressure→**get_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.

pressure→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

pressure→**get_resolution()**

Retourne la résolution des valeurs mesurées.

pressure→**get_unit()**

Retourne l'unité dans laquelle la pression est exprimée.

pressure→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

pressure→**isOnline()**

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

pressure→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

pressure→**load(msValidity)**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

pressure→**loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

pressure→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

pressure→**nextPressure()**

Continue l'énumération des capteurs de pression commencée à l'aide de yFirstPressure().

pressure→**registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

pressure→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

pressure→**set_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour la pression.

pressure→**set_logFrequency(newval)**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

pressure→**set_logicalName(newval)**

Modifie le nom logique du capteur de pression.

pressure→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la pression.

pressure→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

pressure→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

pressure→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

pressure→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPressure.FindPressure() yFindPressure()yFindPressure()

YPressure

Permet de retrouver un capteur de pression d'après un identifiant donné.

```
function yFindPressure( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnline()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de pression sans ambiguïté

Retourne :

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

**YPressure.FirstPressure()
yFirstPressure()yFirstPressure()**

YPressure

Commence l'énumération des capteurs de pression accessibles par la librairie.

```
function yFirstPressure( )
```

Utiliser la fonction `YPressure.nextPressure()` pour itérer sur les autres capteurs de pression.

Retourne :

un pointeur sur un objet `YPressure`, correspondant au premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

pressure→**calibrateFromPoints()**
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.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**describe()****pressure.describe()****YPressure**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de pression (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pressure→**get_advertisedValue()**

YPressure

pressure→**advertisedValue()**

pressure.get_advertisedValue()

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de pression (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

pressure→**get_currentRawValue()**

YPressure

pressure→**currentRawValue()**

pressure.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

pressure→**get_currentValue()**

YPressure

pressure→**currentValue()****pressure.get_currentValue()**

Retourne la mesure actuelle de la pression.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la mesure actuelle de la pression

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

pressure→**get_errorMessage()****YPressure****pressure**→**errorMessage()****pressure.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

pressure→**get_errorType()**

YPressure

pressure→**errorType()****pressure.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

pressure→**get_friendlyName()****YPressure****pressure**→**friendlyName()****pressure.get_friendlyName()**

Retourne un identifiant global du capteur de pression au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de pression si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de pression (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de pression en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pressure→**get_functionDescriptor()**

YPressure

pressure→**functionDescriptor()**

pressure.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

pressure→**get_functionId()****YPressure****pressure**→**functionId()****pressure.get_functionId()**

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de pression (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

pressure→**get_hardwareId()**

YPressure

pressure→**hardwareId()****pressure.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de pression au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de pression (par exemple `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()**

YPressure

pressure→**highestValue()**

pressure.get_highestValue()

Retourne la valeur maximale observée pour la pression.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la pression

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

pressure→**get_logFrequency()**

YPressure

pressure→**logFrequency()**

pressure.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

pressure→**get_logicalName()****YPressure****pressure**→**logicalName()****pressure.get_logicalName()**

Retourne le nom logique du capteur de pression.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de pression. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pressure→**get_lowestValue()**

YPressure

pressure→**lowestValue()****pressure.get_lowestValue()**

Retourne la valeur minimale observée pour la pression.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la pression

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

pressure→**get_module()****YPressure****pressure**→**module()****pressure.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

pressure→**get_module_async()**
pressure→**module_async()**
pressure.get_module_async()

YPressure

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pressure→**get_recordedData()****YPressure****pressure**→**recordedData()****pressure.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez 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()**

YPressure

pressure→**reportFrequency()**

pressure.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

pressure→**get_resolution()****YPressure****pressure**→**resolution()****pressure.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

pressure→**get_unit()**

YPressure

pressure→**unit()****pressure.get_unit()**

Retourne l'unité dans laquelle la pression est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la pression est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

pressure→**get_userdata()****YPressure****pressure**→**userData()****pressure.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

pressure→**isOnline()****pressure.isOnline()**

YPressure

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de pression sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de pression est joignable, `false` sinon

pressure→**isOnline_async()**
pressure.isOnline_async()

YPressure

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de pression sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pressure→load()pressure.load()

YPressure

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**loadCalibrationPoints()**
pressure.loadCalibrationPoints()**YPressure**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**load_async()****pressure.load_async()****YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pressure→**nextPressure()****pressure.nextPressure()****YPressure**

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

```
function nextPressure( )
```

Retourne :

un pointeur sur un objet `YPressure` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pressure→**registerTimedReportCallback()**
pressure.registerTimedReportCallback()

YPressure

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pressure→**set_highestValue()**

YPressure

pressure→**setHighestValue()**

pressure.set_highestValue()

Modifie la mémoire de valeur maximale observée pour la pression.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour la pression

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_logFrequency()****YPressure****pressure**→**setLogFrequency()****pressure.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_logicalName()**

YPressure

pressure→**setLogicalName()**

pressure.set_logicalName()

Modifie le nom logique du capteur de pression.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de pression.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_lowestValue()**
pressure→**setLowestValue()**
pressure.set_lowestValue()

YPressure

Modifie la mémoire de valeur minimale observée pour la pression.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour la pression

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_reportFrequency()**
pressure→**setReportFrequency()**
pressure.set_reportFrequency()

YPressure

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_resolution()****YPressure****pressure**→**setResolution()****pressure.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_userdata()**

YPressure

pressure→**setUserData()****pressure.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

pressure → **wait_async()** **pressure.wait_async()****YPressure**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout :

3.30. Interface de la fonction Pwm

La librairie de programmation Yoctopuce permet simplement de configurer, démarrer et arrêter le PWM.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pwmoutput.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPwmOutput = yoctolib.YPwmOutput;
php	require_once('yocto_pwmoutput.php');
c++	#include "yocto_pwmoutput.h"
m	#import "yocto_pwmoutput.h"
pas	uses yocto_pwmoutput;
vb	yocto_pwmoutput.vb
cs	yocto_pwmoutput.cs
java	import com.yoctopuce.YoctoAPI.YPwmOutput;
py	from yocto_pwmoutput import *

Fonction globales

yFindPwmOutput(func)

Permet de retrouver un PWM d'après un identifiant donné.

yFirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

Méthodes des objets YPwmOutput

pwmoutput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

pwmoutput→dutyCycleMove(target, ms_duration)

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

pwmoutput→get_advertisedValue()

Retourne la valeur courante du PWM (pas plus de 6 caractères).

pwmoutput→get_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

pwmoutput→get_dutyCycleAtPowerOn()

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

pwmoutput→get_enabled()

Retourne l'état de fonctionnement du PWM.

pwmoutput→get_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

pwmoutput→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

pwmoutput→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

pwmoutput→get_frequency()

Retourne la fréquence du PWM en Hz.

pwmoutput→get_friendlyName()

Retourne un identifiant global du PWM au format `NOM_MODULE . NOM_FONCTION`.

pwmoutput→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

`pwmoutput→get_functionId()`

Retourne l'identifiant matériel du PWM, sans référence au module.

`pwmoutput→get_hardwareId()`

Retourne l'identifiant matériel unique du PWM au format `SERIAL.FUNCTIONID`.

`pwmoutput→get_logicalName()`

Retourne le nom logique du PWM.

`pwmoutput→get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`pwmoutput→get_module_async(callback, context)`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`pwmoutput→get_period()`

Retourne la période du PWM en millisecondes.

`pwmoutput→get_pulseDuration()`

Retourne la longueur d'une impulsion du PWM en millisecondes.

`pwmoutput→get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

`pwmoutput→isOnline()`

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

`pwmoutput→isOnline_async(callback, context)`

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

`pwmoutput→load(msValidity)`

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

`pwmoutput→load_async(msValidity, callback, context)`

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

`pwmoutput→nextPwmOutput()`

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

`pwmoutput→pulseDurationMove(ms_target, ms_duration)`

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

`pwmoutput→registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

`pwmoutput→set_dutyCycle(newval)`

Modifie le duty cycle du PWM, en pour cents.

`pwmoutput→set_dutyCycleAtPowerOn(newval)`

Modifie le duty cycle du PWM au démarrage du module.

`pwmoutput→set_enabled(newval)`

Démarre ou arrête le PWM.

`pwmoutput→set_enabledAtPowerOn(newval)`

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

`pwmoutput→set_frequency(newval)`

Modifie la fréquence du PWM.

`pwmoutput→set_logicalName(newval)`

Modifie le nom logique du PWM.

`pwmoutput→set_period(newval)`

Modifie la période du PWM.

3. Reference

pwmoutput→**set_pulseDuration(newval)**

Modifie la longueur des impulsions du PWM, en millisecondes.

pwmoutput→**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`.

pwmoutput→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPwmOutput.FindPwmOutput() yFindPwmOutput(yFindPwmOutput())

YPwmOutput

Permet de retrouver un PWM d'après un identifiant donné.

```
function yFindPwmOutput( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le PWM soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmOutput.isOnline()` pour tester si le PWM est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le PWM sans ambiguïté

Retourne :

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

YPwmOutput.FirstPwmOutput() yFirstPwmOutput()yFirstPwmOutput()

YPwmOutput

Commence l'énumération des PWM accessibles par la librairie.

```
function yFirstPwmOutput( )
```

Utiliser la fonction `YPwmOutput.nextPwmOutput()` pour itérer sur les autres PWM.

Retourne :

un pointeur sur un objet `YPwmOutput`, correspondant au premier PWM accessible en ligne, ou `null` si il n'y a pas de PWM disponibles.

pwmoutput→describe()**pwmoutput.describe()****YPwmOutput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le PWM (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pwmoutput→dutyCycleMove()
pwmoutput.dutyCycleMove()

YPwmOutput

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

```
function dutyCycleMove( target, ms_duration)
```

Paramètres :

target nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)
ms_duration durée totale de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→get_advertisedValue()

YPwmOutput

pwmoutput→advertisedValue()

pwmoutput.get_advertisedValue()

Retourne la valeur courante du PWM (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du PWM (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

pwmoutput→**get_dutyCycle()**

YPwmOutput

pwmoutput→**dutyCycle()****pwmoutput.get_dutyCycle()**

Retourne le duty cycle du PWM, en pour cents.

```
function get_dutyCycle( )
```

Retourne :

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne `Y_DUTYCYCLE_INVALID`.

pwmoutput→get_dutyCycleAtPowerOn()**YPwmOutput****pwmoutput→dutyCycleAtPowerOn()****pwmoutput.get_dutyCycleAtPowerOn()**

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

```
function get_dutyCycleAtPowerOn( )
```

Retourne :

une valeur numérique représentant le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

En cas d'erreur, déclenche une exception ou retourne `Y_DUTYCYCLEATPOWERON_INVALID`.

pwmoutput→get_enabled()

YPwmOutput

pwmoutput→enabled()pwmoutput.get_enabled()

Retourne l'état de fonctionnement du PWM.

function **get_enabled()**

Retourne :

soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon l'état de fonctionnement du PWM

En cas d'erreur, déclenche une exception ou retourne Y_ENABLED_INVALID.

pwmoutput→get_enabledAtPowerOn()

YPwmOutput

pwmoutput→enabledAtPowerOn()

pwmoutput.get_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

```
function get_enabledAtPowerOn( )
```

Retourne :

soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état de fonctionnement du PWM à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLEDATPOWERON_INVALID`.

`pwmoutput→get_errorMessage()`

YPwmOutput

`pwmoutput→errorMessage()`

`pwmoutput.get_errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

pwmoutput→**get_errorType()****YPwmOutput****pwmoutput**→**errorType()****pwmoutput.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

`pwmoutput`→`get_frequency()`

`YPwmOutput`

`pwmoutput`→`frequency()``pwmoutput.get_frequency()`

Retourne la fréquence du PWM en Hz.

```
function get_frequency() ( )
```

Retourne :

un entier représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne `Y_FREQUENCY_INVALID`.

pwmoutput→**get_friendlyName()****YPwmOutput****pwmoutput**→**friendlyName()****pwmoutput.get_friendlyName()**

Retourne un identifiant global du PWM au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du PWM si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du PWM (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le PWM en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pwmoutput→get_functionDescriptor()
pwmoutput→functionDescriptor()
pwmoutput.get_functionDescriptor()

YPwmOutput

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

pwmoutput→**get_functionId()****YPwmOutput****pwmoutput**→**functionId()****pwmoutput.get_functionId()**

Retourne l'identifiant matériel du PWM, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le PWM (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

`pwmoutput→get_hardwareId()`

YPwmOutput

`pwmoutput→hardwareId()`

`pwmoutput.get_hardwareId()`

Retourne l'identifiant matériel unique du PWM au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du PWM (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le PWM (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pwmoutput→get_logicalName()
pwmoutput→logicalName()
pwmoutput.get_logicalName()

YPwmOutput

Retourne le nom logique du PWM.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du PWM. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pwmoutput→get_module()

YPwmOutput

pwmoutput→module()pwmoutput.get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

pwmoutput→**get_module_async()****YPwmOutput****pwmoutput**→**module_async()****pwmoutput.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pwmoutput→**get_period()**

YPwmOutput

pwmoutput→**period()****pwmoutput.get_period()**

Retourne la période du PWM en millisecondes.

```
function get_period( )
```

Retourne :

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne `Y_PERIOD_INVALID`.

pwmoutput→get_pulseDuration()

YPwmOutput

pwmoutput→pulseDuration()

pwmoutput.get_pulseDuration()

Retourne la longueur d'une impulsion du PWM en millisecondes.

```
function get_pulseDuration( )
```

Retourne :

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y_PULSEDURATION_INVALID.

pwmoutput→get_userData()

YPwmOutput

pwmoutput→userData()pwmoutput.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

pwmoutput→**isOnline()****pwmoutput.isOnline()****YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du PWM sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le PWM est joignable, `false` sinon

pwmoutput→isOnline_async() pwmoutput.isOnline_async()

YPwmOutput

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du PWM sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pwmoutput→**load()****pwmoutput.load()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→load_async()**pwmoutput.load_async()****YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pwmoutput→**nextPwmOutput()**
pwmoutput.nextPwmOutput()

YPwmOutput

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

```
function nextPwmOutput()
```

Retourne :

un pointeur sur un objet `YPwmOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pwmoutput→**pulseDurationMove()**
pwmoutput.pulseDurationMove()**YPwmOutput**

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

```
function pulseDurationMove( ms_target, ms_duration)
```

N'importe quel changement de fréquence, duty cycle, période ou encore de longueur d'impulsion annulera tout processus de transition en cours.

Paramètres :

- ms_target** nouvelle longueur des impulsions à la fin de la transition (nombre flottant, représentant la longueur en millisecondes)
- ms_duration** durée totale de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→**registerValueCallback()**
pwmoutput.registerValueCallback()

YPwmOutput

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pwmoutput→set_dutyCycle()
pwmoutput→setDutyCycle()
pwmoutput.set_dutyCycle()

YPwmOutput

Modifie le duty cycle du PWM, en pour cents.

```
function set_dutyCycle( newval)
```

Paramètres :

newval une valeur numérique représentant le duty cycle du PWM, en pour cents

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→**set_dutyCycleAtPowerOn()**
pwmoutput→**setDutyCycleAtPowerOn()**
pwmoutput.set_dutyCycleAtPowerOn()

YPwmOutput

Modifie le duty cycle du PWM au démarrage du module.

```
function set_dutyCycleAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur numérique représentant le duty cycle du PWM au démarrage du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_enabled()

YPwmOutput

pwmoutput→setEnabled()pwmoutput.set_enabled()

Démarre ou arrête le PWM.

```
function set_enabled( newval)
```

Paramètres :

newval soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`pwmoutput`→`set_enabledAtPowerOn()`
`pwmoutput`→`setEnabledAtPowerOn()`
`pwmoutput.set_enabledAtPowerOn()`

YPwmOutput

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

```
function set_enabledAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du fonctionnement du PWM à la mise sous tension du module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`pwmoutput→set_frequency()`
`pwmoutput→setFrequency()`
`pwmoutput.set_frequency()`

YPwmOutput

Modifie la fréquence du PWM.

```
function set_frequency( newval)
```

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

Paramètres :

`newval` un entier représentant la fréquence du PWM

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→**set_logicalName()**
pwmoutput→**setLogicalName()**
pwmoutput.set_logicalName()

YPwmOutput

Modifie le nom logique du PWM.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du PWM.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`pwmoutput`→`set_period()`

YPwmOutput

`pwmoutput`→`setPeriod()``pwmoutput.set_period()`

Modifie la période du PWM.

```
function set_period( newval)
```

Paramètres :

newval une valeur numérique représentant la période du PWM

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_pulseDuration()
pwmoutput→setPulseDuration()
pwmoutput.set_pulseDuration()

YPwmOutput

Modifie la longueur des impulsion du PWM, en millisecondes.

```
function set_pulseDuration( newval)
```

Attention la longueur d'un impulsion ne peut pas être plus grande que la période, dans la cas contraire, la longueur sera automatiquement tronqué à la période.

Paramètres :

newval une valeur numérique représentant la longueur des impulsion du PWM, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_userdata()
pwmoutput→setUserData()
pwmoutput.set_userdata()

YPwmOutput

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

pwmoutput→**wait_async()****pwmoutput.wait_async()****YPwmOutput**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.31. Interface de la fonction PwmPowerSource

La librairie de programmation Yoctopuce permet de configurer la source de tension utilisée par tous les PWM situés sur un même module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pwmpowersource.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPwmPowerSource = yoctolib.YPwmPowerSource;
php	require_once('yocto_pwmpowersource.php');
c++	#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(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`.

`pwmpowersource→wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPwmPowerSource.FindPwmPowerSource() yFindPwmPowerSource()yFindPwmPowerSource()

YPwmPowerSource

Permet de retrouver une source de tension d'après un identifiant donné.

```
function yFindPwmPowerSource( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmPowerSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence la source de tension sans ambiguïté

Retourne :

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

**YPwmPowerSource.FirstPwmPowerSource()
yFirstPwmPowerSource(yFirstPwmPowerSource())**

YPwmPowerSource

Commence l'énumération des Source de tension accessibles par la librairie.

```
function yFirstPwmPowerSource( )
```

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource()` pour itérer sur les autres Source de tension.

Retourne :

un pointeur sur un objet `YPwmPowerSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de Source de tension disponibles.

pwmpowersource→**describe()**
pwmpowersource.describe()**YPwmPowerSource**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

```
une chaîne de caractères décrivant la source de tension (ex:  
Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)
```

`pwmpowersource→get_advertisedValue()`

YPwmPowerSource

`pwmpowersource→advertisedValue()`

`pwmpowersource.get_advertisedValue()`

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

`pwmpowersource`→`get_errorMessage()`

YPwmPowerSource

`pwmpowersource`→`errorMessage()`

`pwmpowersource`.`get_errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

`pwmpowersource→get_errorType()`

YPwmPowerSource

`pwmpowersource→errorType()`

`pwmpowersource.get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

pwmpowersource→**get_friendlyName()**

YPwmPowerSource

pwmpowersource→**friendlyName()**

pwmpowersource.get_friendlyName()

Retourne un identifiant global de la source de tension au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la source de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la source de tension (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la source de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pwmpowersource→**get_functionDescriptor()**
pwmpowersource→**functionDescriptor()**
pwmpowersource.get_functionDescriptor()

YPwmPowerSource

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

`pwmpowersource`→`get_functionId()`

YPwmPowerSource

`pwmpowersource`→`functionId()`

`pwmpowersource.get_functionId()`

Retourne l'identifiant matériel de la source de tension, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la source de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

`pwmpowersource`→`get_hardwareId()`

YPwmPowerSource

`pwmpowersource`→`hardwareId()`

`pwmpowersource.get_hardwareId()`

Retourne l'identifiant matériel unique de la source de tension au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la source de tension (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant la source de tension (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pwmpowersource→get_logicalName()

YPwmPowerSource

pwmpowersource→logicalName()

pwmpowersource.get_logicalName()

Retourne le nom logique de la source de tension.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de la source de tension. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pwmpowersource→get_module()

YPwmPowerSource

pwmpowersource→module()

pwmpowersource.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

`pwmpowersource`→`get_module_async()`

`YPwmPowerSource`

`pwmpowersource`→`module_async()`

`pwmpowersource.get_module_async()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pwmpowersource→**get_powerMode()**

YPwmPowerSource

pwmpowersource→**powerMode()**

pwmpowersource.get_powerMode()

Retourne la source de tension utilisé par tous les PWM du même module.

```
function get_powerMode( )
```

Retourne :

une valeur parmi Y_POWERMODE_USB_5V, Y_POWERMODE_USB_3V, Y_POWERMODE_EXT_V et Y_POWERMODE_OPNDRN représentant la source de tension utilisé par tous les PWM du même module

En cas d'erreur, déclenche une exception ou retourne Y_POWERMODE_INVALID.

pwmpowersource→get_userData()

YPwmPowerSource

pwmpowersource→userData()

pwmpowersource.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

**pwmpowersource→isOnline()
pwmpowersource.isOnline()**

YPwmPowerSource

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de la source de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si la source de tension est joignable, `false` sinon

pwmpowersource→**isOnline_async()**
pwmpowersource.isOnline_async()

YPwmPowerSource

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de la source de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pwmpowersource→load()pwmpowersource.load()**YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→**load_async()**
pwmpowersource.load_async()**YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

pwmpowersource→**nextPwmPowerSource()**
pwmpowersource.nextPwmPowerSource()

YPwmPowerSource

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

```
function nextPwmPowerSource( )
```

Retourne :

un pointeur sur un objet `YPwmPowerSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pwmpowersource→**registerValueCallback()**
pwmpowersource.registerValueCallback()

YPwmPowerSource

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pwmpowersource→set_logicalName()**YPwmPowerSource****pwmpowersource→setLogicalName()****pwmpowersource.set_logicalName()**

Modifie le nom logique de la source de tension.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la source de tension.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→**set_powerMode()**
pwmpowersource→**setPowerMode()**
pwmpowersource.set_powerMode()

YPwmPowerSource

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

```
function set_powerMode( newval)
```

Le PWM peut aussi en mode open drain, dans ce code il tire activement la ligne à zéro volts. Attention ce paramètre est commun à tous les PWM du module, si vous changez le valeur de ce paramètre, tous les PWM situés sur le même module seront affectés. Si vous souhaitez que le changement de ce paramètre soit conservé après un redémarrage du module, n'oubliez pas d'appeler la méthode `saveToFlash()`.

Paramètres :

newval une valeur parmi `Y_POWERMODE_USB_5V`, `Y_POWERMODE_USB_3V`, `Y_POWERMODE_EXT_V` et `Y_POWERMODE_OPNDRN` représentant le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→**set_userdata()****YPwmPowerSource****pwmpowersource**→**setUserData()****pwmpowersource.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

pwmpowersource→**wait_async()**
pwmpowersource.wait_async()

YPwmPowerSource

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.32. Interface du quaternion

La class YQt de la librairie Yoctopuce permet d'accéder à l'estimation de l'orientation tridimensionnelle du Yocto-3D sous forme d'un quaternion. Il n'est en général pas nécessaire d'y accéder directement, la classe YGyro offrant une abstraction de plus haut niveau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_gyro.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YGyro = yoctolib.YGyro;</code>
php	<code>require_once('yocto_gyro.php');</code>
c++	<code>#include "yocto_gyro.h"</code>
m	<code>#import "yocto_gyro.h"</code>
pas	<code>uses yocto_gyro;</code>
vb	<code>yocto_gyro.vb</code>
cs	<code>yocto_gyro.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YGyro;</code>
py	<code>from yocto_gyro import *</code>

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 retournée par le capteur (sans arrondi ni calibration).

qt→get_currentValue()

Retourne la valeur actuelle de la coordonnée.

qt→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

qt→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

qt→get_friendlyName()

Retourne un identifiant global de l'élément de quaternion au format `NOM_MODULE . NOM_FONCTION`.

qt→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

qt→get_functionId()

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

3. Reference

qt→get_hardwareId()

Retourne l'identifiant matériel unique de l'élément de quaternion au format SERIAL . FUNCTIONID.

qt→get_highestValue()

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

qt→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

qt→get_logicalName()

Retourne le nom logique de l'élément de quaternion.

qt→get_lowestValue()

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

qt→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

qt→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

qt→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.

qt→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

qt→get_resolution()

Retourne la résolution des valeurs mesurées.

qt→get_unit()

Retourne l'unité dans laquelle la coordonnée est exprimée.

qt→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

qt→isOnline()

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

qt→isOnline_async(callback, context)

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

qt→load(msValidity)

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

qt→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

qt→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

qt→nextQt()

Continue l'énumération des éléments de quaternion commencée à l'aide de yFirstQt().

qt→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

qt→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

qt→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

qt→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

qt→set_logicalName(newval)

Modifie le nom logique de l'élément de quaternion.

qt→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

qt→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

qt→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

qt→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

qt→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YQt.FindQt() yFindQt(yFindQt())

YQt

Permet de retrouver un élément de quaternion d'après un identifiant donné.

```
function yFindQt( func )
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnline()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

Retourne :

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

YQt.FirstQt()
yFirstQt()yFirstQt()

YQt

Commence l'énumération des éléments de quaternion accessibles par la librairie.

```
function yFirstQt( )
```

Utiliser la fonction `YQt.nextQt()` pour itérer sur les autres éléments de quaternion.

Retourne :

un pointeur sur un objet `YQt`, correspondant au premier élément de quaternion accessible en ligne, ou `null` si il n'y a pas de éléments de quaternion disponibles.

qt→calibrateFromPoints()qt.calibrateFromPoints()

YQt

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→describe()qt.describe()**YQt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'élément de quaternion (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

qt→get_advertisedValue()

YQt

qt→advertisedValue()qt.get_advertisedValue()

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

qt→get_currentRawValue()**YQt****qt→currentRawValue()qt.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

qt→**get_currentValue()**

YQt

qt→**currentValue()****qt.get_currentValue()**

Retourne la valeur actuelle de la coordonnée.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de la coordonnée

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

qt→get_errorMessage()

YQt

qt→errorMessage()qt.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

qt→get_errorType()

YQt

qt→errorType()qt.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

qt→**get_friendlyName()**

YQt

qt→**friendlyName()****qt.get_friendlyName()**

Retourne un identifiant global de l'élément de quaternion au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'élément de quaternion si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'élément de quaternion (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'élément de quaternion en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

qt→**get_functionDescriptor()**

YQt

qt→**functionDescriptor()****qt.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

qt→**get_functionId()**

YQt

qt→**functionId()****qt.get_functionId()**

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'élément de quaternion (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

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( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'élément de quaternion (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'élément de quaternion (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

qt→**get_highestValue()**

YQt

qt→**highestValue()****qt.get_highestValue()**

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

qt→get_logFrequency()

YQt

qt→logFrequency()qt.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

qt→**get_logicalName()**

YQt

qt→**logicalName()****qt.get_logicalName()**

Retourne le nom logique de l'élément de quaternion.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'élément de quaternion. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

qt→**get_lowestValue()**

YQt

qt→**lowestValue()****qt.get_lowestValue()**

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

qt→get_module()**YQt****qt→module()qt.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

qt→**get_module_async()**

YQt

qt→**module_async()****qt.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

qt→**get_recordedData()**

YQt

qt→**recordedData()****qt.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

qt→**get_reportFrequency()**

YQt

qt→**reportFrequency()****qt.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

qt→get_resolution()

YQt

qt→resolution()qt.get_resolution()

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

qt→**get_unit()**

YQt

qt→**unit()****qt.get_unit()**

Retourne l'unité dans laquelle la coordonnée est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la coordonnée est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

qt→**get_userdata()**

YQt

qt→**userData()****qt.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es 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.

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'élément de quaternion est joignable, `false` sinon

qt→isOnline_async()qt.isOnline_async()**YQt**

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

qt→load()qt.load()

YQt

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→loadCalibrationPoints()qt.loadCalibrationPoints()**YQt**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→load_async()qt.load_async()

YQt

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

qt→**nextQt()****qt.nextQt()****YQt**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

```
function nextQt( )
```

Retourne :

un pointeur sur un objet `YQt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

qt→registerTimedReportCallback()
qt.registerTimedReportCallback()

YQt

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

qt→**set_highestValue()**

YQt

qt→**setHighestValue()****qt.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_logFrequency()

YQt

qt→setLogFrequency()qt.set_logFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→**set_logicalName()**

YQt

qt→**setLogicalName()****qt.set_logicalName()**

Modifie le nom logique de l'élément de quaternion.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'élément de quaternion.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→**set_lowestValue()**

YQt

qt→**setLowestValue()****qt.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→**set_reportFrequency()**

YQt

qt→**setReportFrequency()****qt.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→**set_resolution()**

YQt

qt→**setResolution()****qt.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→**set_userdata()**

YQt

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`.

```
function set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

qt→**wait_async()****qt.wait_async()****YQt**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.33. Interface de la fonction Horloge Temps Real

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	<code><script type='text/javascript' src='yocto_realtimelock.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YRealTimeClock = yoctolib.YRealTimeClock;</code>
php	<code>require_once('yocto_realtimelock.php');</code>
c++	<code>#include "yocto_realtimelock.h"</code>
m	<code>#import "yocto_realtimelock.h"</code>
pas	<code>uses yocto_realtimelock;</code>
vb	<code>yocto_realtimelock.vb</code>
cs	<code>yocto_realtimelock.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YRealTimeClock;</code>
py	<code>from yocto_realtimelock import *</code>

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

realtimelock→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

realtimelock→get_advertisedValue()

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

realtimelock→get_dateTime()

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

realtimelock→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

realtimelock→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

realtimelock→get_friendlyName()

Retourne un identifiant global de l'horloge au format `NOM_MODULE . NOM_FONCTION`.

realtimelock→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

realtimelock→get_functionId()

Retourne l'identifiant matériel de l'horloge, sans référence au module.

realtimelock→get_hardwareId()

Retourne l'identifiant matériel unique de l'horloge au format `SERIAL . FUNCTIONID`.

realtimelock→get_logicalName()

Retourne le nom logique de l'horloge.

realtimelock→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`realtimeclock→get_module_async(callback, context)`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`realtimeclock→get_timeSet()`

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

`realtimeclock→get_unixTime()`

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

`realtimeclock→get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

`realtimeclock→get_utcOffset()`

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

`realtimeclock→isOnline()`

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

`realtimeclock→isOnline_async(callback, context)`

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

`realtimeclock→load(msValidity)`

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

`realtimeclock→load_async(msValidity, callback, context)`

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

`realtimeclock→nextRealTimeClock()`

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

`realtimeclock→registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

`realtimeclock→set_logicalName(newval)`

Modifie le nom logique de l'horloge.

`realtimeclock→set_unixTime(newval)`

Modifie l'heure courante.

`realtimeclock→set_userData(data)`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

`realtimeclock→set_utcOffset(newval)`

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

`realtimeclock→wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRealTimeClock.FindRealTimeClock() yFindRealTimeClock()yFindRealTimeClock()

YRealTimeClock

Permet de retrouver une horloge d'après un identifiant donné.

```
function yFindRealTimeClock( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnline()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'horloge sans ambiguïté

Retourne :

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

**YRealTimeClock.FirstRealTimeClock()
yFirstRealTimeClock()yFirstRealTimeClock()**

YRealTimeClock

Commence l'énumération des horloge accessibles par la librairie.

```
function yFirstRealTimeClock( )
```

Utiliser la fonction `YRealTimeClock.nextRealTimeClock()` pour itérer sur les autres horloge.

Retourne :

un pointeur sur un objet `YRealTimeClock`, correspondant à la première horloge accessible en ligne, ou `null` si il n'y a pas de horloge disponibles.

realtimeclock→describe()realtimeclock.describe()

YRealTimeClock

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format `TYPE (NAME) =SERIAL .FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'horloge (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

realtimeclock→**get_advertisedValue()****YRealTimeClock****realtimeclock**→**advertisedValue()****realtimeclock.get_advertisedValue()**

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

`realtimeclock→get_dateTime()`

YRealTimeClock

`realtimeclock→dateTime()`

`realtimeclock.get_dateTime()`

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

```
function get_dateTime( )
```

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()**YRealTimeClock****realtimeclock→errorMessage()****realtimeclock.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

realtimeclock→get_errorType()

YRealTimeClock

realtimeclock→errorType()

realtimeclock.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

realtimeclock→get_friendlyName()**YRealTimeClock****realtimeclock→friendlyName()****realtimeclock.get_friendlyName()**

Retourne un identifiant global de l'horloge au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'horloge si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'horloge (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'horloge en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

realtimeclock→get_functionDescriptor()
realtimeclock→functionDescriptor()
realtimeclock.get_functionDescriptor()

YRealTimeClock

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

realtimeclock→get_functionId()
realtimeclock→functionId()
realtimeclock.get_functionId()

YRealTimeClock

Retourne l'identifiant matériel de l'horloge, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'horloge (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

realtimeclock→get_hardwareId()
realtimeclock→hardwareId()
realtimeclock.get_hardwareId()

YRealTimeClock

Retourne l'identifiant matériel unique de l'horloge au format SERIAL . FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'horloge (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'horloge (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

realtimeclock→**get_logicalName()****YRealTimeClock****realtimeclock**→**logicalName()****realtimeclock.get_logicalName()**

Retourne le nom logique de l'horloge.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'horloge. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

realtimeclock→get_module()

YRealTimeClock

realtimeclock→module()realtimeclock.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

realtimeclock→**get_module_async()****YRealTimeClock****realtimeclock**→**module_async()****realtimeclock.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

realtimeclock→get_timeSet()

YRealTimeClock

realtimeclock→timeSet()realtimeclock.get_timeSet()

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

```
function get_timeSet( )
```

Retourne :

soit Y_TIMESET_FALSE, soit Y_TIMESET_TRUE, selon vrai si l'horloge à été mise à l'heure, sinon faux

En cas d'erreur, déclenche une exception ou retourne Y_TIMESET_INVALID.

realtimeclock→get_unixTime()**YRealTimeClock****realtimeclock→unixTime()****realtimeclock.get_unixTime()**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

```
function get_unixTime( )
```

Retourne :

un entier représentant l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970)

En cas d'erreur, déclenche une exception ou retourne `Y_UNIXTIME_INVALID`.

realtimeclock→get_userdata()

YRealTimeClock

realtimeclock→userdata()

realtimeclock.get_userdata()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

realtimeclock→get_utcOffset()**YRealTimeClock****realtimeclock→utcOffset()****realtimeclock.get_utcOffset()**

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
function get_utcOffset( )
```

Retourne :

un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne `Y_UTCOffset_INVALID`.

realtimeclock→isOnline()**realtimeclock.isOnline()**

YRealTimeClock

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'horloge est joignable, `false` sinon

realtimeclock→isOnline_async()
realtimeclock.isOnline_async()

YRealTimeClock

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

realtimeclock→load()realtimeclock.load()

YRealTimeClock

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

**realtimeclock→load_async()
realtimeclock.load_async()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

realtimeclock→**nextRealTimeClock()**
realtimeclock.nextRealTimeClock()

YRealTimeClock

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

```
function nextRealTimeClock( )
```

Retourne :

un pointeur sur un objet `YRealTimeClock` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**realtimelock→registerValueCallback()
realtimelock.registerValueCallback()**

YRealTimeClock

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

realtimeclock→set_logicalName()
realtimeclock→setLogicalName()
realtimeclock.set_logicalName()

YRealTimeClock

Modifie le nom logique de l'horloge.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'horloge.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`realtimeclock`→`set_unixTime()`
`realtimeclock`→`setUnixTime()`
`realtimeclock.set_unixTime()`

YRealTimeClock

Modifie l'heure courante.

```
function set_unixTime( newval)
```

L'heure est passée au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970). Si l'heure UTC est connue, l'attribut `utcOffset` sera automatiquement ajusté en fonction de l'heure configurée.

Paramètres :

newval un entier représentant l'heure courante

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→set_userdata()
realtimeclock→setUserData()
realtimeclock.set_userdata()

YRealTimeClock

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

`realtimeclock`→`set_utcOffset()`
`realtimeclock`→`setUtcOffset()`
`realtimeclock.set_utcOffset()`

YRealTimeClock

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
function set_utcOffset( newval)
```

Le décalage 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.

realtimelock→**wait_async()**
realtimelock.wait_async()

YRealTimeClock

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.34. Configuration du référentiel

Cette classe permet de configurer l'orientation dans laquelle le Yocto-3D est utilisé, afin que les fonctions d'orientation relatives au plan de la surface terrestre utilisent le référentiel approprié. La classe offre aussi un processus de recalibration tridimensionnel des capteurs, permettant de compenser les variations locales de l'accélération terrestre et d'améliorer la précision des capteurs d'inclinaisons.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_refframe.js'></script>
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.

reiframe→**get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

reiframe→**get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

reiframe→**get_friendlyName()**

Retourne un identifiant global du référentiel au format NOM_MODULE . NOM_FONCTION.

reiframe→**get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

reiframe→**get_functionId()**

Retourne l'identifiant matériel du référentiel, sans référence au module.

reiframe→**get_hardwareId()**

Retourne l'identifiant matériel unique du référentiel au format SERIAL . FUNCTIONID.

reiframe→**get_logicalName()**

Retourne le nom logique du référentiel.

reiframe→**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

reiframe→**get_module_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

reiframe→**get_mountOrientation()**

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

reiframe→**get_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

reiframe→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

reiframe→**isOnline()**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

reiframe→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

reiframe→**load(msValidity)**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

reiframe→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

reiframe→**more3DCalibration()**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode start3DCalibration.

reiframe→**nextRefFrame()**

Continue l'énumération des référentiels commencée à l'aide de yFirstRefFrame().

reiframe→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

reiframe→**save3DCalibration()**

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

reiframe→**set_bearing(newval)**

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(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`.

refframe→**start3DCalibration()**

Initie le processus de calibration tridimensionnelle des capteurs.

refframe→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRefFrame.FindRefFrame() yFindRefFrame()yFindRefFrame()

YRefFrame

Permet de retrouver un référentiel d'après un identifiant donné.

```
function yFindRefFrame( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRefFrame.isOnline()` pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le référentiel sans ambiguïté

Retourne :

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

**YRefFrame.FirstRefFrame()
yFirstRefFrame(yFirstRefFrame())**

YRefFrame

Commence l'énumération des référentiels accessibles par la librairie.

```
function yFirstRefFrame( )
```

Utiliser la fonction `YRefFrame.nextRefFrame()` pour itérer sur les autres référentiels.

Retourne :

un pointeur sur un objet `YRefFrame`, correspondant au premier référentiel accessible en ligne, ou `null` si il n'y a pas de référentiels disponibles.

refframe→**cancel3DCalibration()**
refframe.cancel3DCalibration()

YRefFrame

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

```
function cancel3DCalibration( )
```

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→describe()refframe.describe()**YRefFrame**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le référentiel (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

refframe→**get_3DCalibrationHint()**

YRefFrame

refframe→**3DCalibrationHint()**

refframe.get_3DCalibrationHint()

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

```
function get_3DCalibrationHint( )
```

Retourne :

une chaîne de caractères.

refframe→**get_3DCalibrationLogMsg()**
refframe→**3DCalibrationLogMsg()**
refframe.get_3DCalibrationLogMsg()

YRefFrame

Retourne le dernier message de log produit par le processus de calibration.

```
function get_3DCalibrationLogMsg( )
```

Si aucun nouveau message n'est disponible, retourne une chaîne vide.

Retourne :

une chaîne de caractères.

refframe→get_3DCalibrationProgress()

YRefFrame

refframe→3DCalibrationProgress()

refframe.get_3DCalibrationProgress()

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.

```
function get_3DCalibrationProgress( )
```

Retourne :

une nombre entier entre 0 (pas commencé) et 100 (terminé).

refframe→**get_3DCalibrationStage()****YRefFrame****refframe**→**3DCalibrationStage()****refframe.get_3DCalibrationStage()**

Retourne l'index de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

```
function get_3DCalibrationStage( )
```

Retourne :

une nombre entier, croissant au fur et à mesure de la complétion des étapes.

refframe→get_3DCalibrationStageProgress()

YRefFrame

refframe→3DCalibrationStageProgress()

refframe.get_3DCalibrationStageProgress()

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

```
function get_3DCalibrationStageProgress( )
```

Retourne :

une nombre entier entre 0 (pas commencé) et 100 (terminé).

refframe→**get_advertisedValue()****YRefFrame****refframe**→**advertisedValue()****refframe.get_advertisedValue()**

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du référentiel (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

refframe→**get_bearing()**

YRefFrame

refframe→**bearing()****refframe.get_bearing()**

Retourne le cap de référence utilisé par le compas.

```
function get_bearing( )
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici.

Retourne :

une valeur numérique représentant le cap de référence utilisé par le compas

En cas d'erreur, déclenche une exception ou retourne `Y_BEARING_INVALID`.

refframe→**get_errorMessage()****YRefFrame****refframe**→**errorMessage()****refframe.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

refframe→**get_errorType()**

YRefFrame

refframe→**errorType()****refframe.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

refframe→**get_friendlyName()****YRefFrame****refframe**→**friendlyName()****refframe.get_friendlyName()**

Retourne un identifiant global du référentiel au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du référentiel si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du référentiel (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le référentiel en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

refframe→**get_functionDescriptor()**
refframe→**functionDescriptor()**
refframe.get_functionDescriptor()

YRefFrame

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

refframe→**get_functionId()****YRefFrame****refframe**→**functionId()****refframe.get_functionId()**

Retourne l'identifiant matériel du référentiel, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le référentiel (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

refframe→get_hardwareId()

YRefFrame

refframe→hardwareId()refframe.get_hardwareId()

Retourne l'identifiant matériel unique du référentiel au format SERIAL . FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du référentiel (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le référentiel (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

refframe→**get_logicalName()****YRefFrame****refframe**→**logicalName()****refframe.get_logicalName()**

Retourne le nom logique du référentiel.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du référentiel. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

refframe→**get_module()**

YRefFrame

refframe→**module()**refframe.**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

refframe→**get_module_async()****YRefFrame****refframe**→**module_async()****refframe.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

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( )
```

Retourne :

une valeur parmi l'énumération `Y_MOUNTORIENTATION` (`Y_MOUNTORIENTATION_TWELVE`, `Y_MOUNTORIENTATION_THREE`, `Y_MOUNTORIENTATION_SIX`, `Y_MOUNTORIENTATION_NINE`) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face `BOTTOM` le 12h pointe vers l'avant, tandis que sur la face `TOP` le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**get_mountPosition()****YRefFrame****refframe**→**mountPosition()****refframe.get_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

```
function get_mountPosition( )
```

Retourne :

une valeur parmi l'énumération `Y_MOUNTPOSITION` (`Y_MOUNTPOSITION_BOTTOM`, `Y_MOUNTPOSITION_TOP`, `Y_MOUNTPOSITION_FRONT`, `Y_MOUNTPOSITION_RIGHT`, `Y_MOUNTPOSITION_REAR`, `Y_MOUNTPOSITION_LEFT`), correspondant à l'installation dans une boîte, sur l'une des six faces

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→get_userData()

YRefFrame

refframe→userData()refframe.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

refframe→**isOnline()****refframe.isOnline()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du référentiel sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le référentiel est joignable, `false` sinon

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du référentiel sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

refframe→**load()****refframe.load()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**load_async()****refframe.load_async()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

refframe→**more3DCalibration()**
refframe.more3DCalibration()**YRefFrame**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

```
function more3DCalibration( )
```

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode `get_3DCalibrationHint` (les instructions changent pendant la procédure de calibration). En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**nextRefFrame()****refframe.nextRefFrame()**

YRefFrame

Continue l'énumération des référentiels commencée à l'aide de `yFirstRefFrame()`.

```
function nextRefFrame( )
```

Retourne :

un pointeur sur un objet `YRefFrame` accessible en ligne, ou `null` lorsque l'énumération est terminée.

refframe→**registerValueCallback()**
refframe.registerValueCallback()

YRefFrame

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

refframe→**save3DCalibration()**
refframe.save3DCalibration()

YRefFrame

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

```
function save3DCalibration( )
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après le redémarrage du module. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**set_bearing()****YRefFrame****refframe**→**setBearing()****refframe.set_bearing()**

Modifie le cap de référence utilisé par le compas.

```
function set_bearing( newval)
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici. Par exemple, si vous indiquez comme cap de référence la valeur de la déclinaison magnétique terrestre, le compas donnera l'orientation par rapport au Nord géographique. De même, si le capteur n'est pas positionné dans une des directions standard à cause d'un angle de lacet supplémentaire, vous pouvez le configurer comme cap de référence afin que le compas donne la direction naturelle attendue.

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant le cap de référence utilisé par le compas

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**set_logicalName()**

YRefFrame

refframe→**setLogicalName()**

refframe.set_logicalName()

Modifie le nom logique du référentiel.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du référentiel.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**set_mountPosition()****YRefFrame****refframe**→**setMountPosition()****refframe.set_mountPosition()**

Modifie le référentiel de la boussole et des inclinomètres.

```
function set_mountPosition( position, orientation)
```

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas où le module n'est pas utilisé horizontalement et à l'endroit, il faut indiquer son orientation de référence (parallèle à la surface terrestre) afin que les mesures soient faites relativement à cette position.

Paramètres :

position une valeur parmi l'énumération `Y_MOUNTPOSITION` (`Y_MOUNTPOSITION_BOTTOM`, `Y_MOUNTPOSITION_TOP`, `Y_MOUNTPOSITION_FRONT`, `Y_MOUNTPOSITION_RIGHT`, `Y_MOUNTPOSITION_REAR`, `Y_MOUNTPOSITION_LEFT`), correspondant à l'installation dans une boîte, sur l'une des six faces.

orientation une valeur parmi l'énumération `Y_MOUNTORIENTATION` (`Y_MOUNTORIENTATION_TWELVE`, `Y_MOUNTORIENTATION_THREE`, `Y_MOUNTORIENTATION_SIX`, `Y_MOUNTORIENTATION_NINE`) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face `BOTTOM` le 12h pointe vers l'avant, tandis que sur la face `TOP` le 12h pointe vers l'arrière. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

refframe→**set_userdata()**

YRefFrame

refframe→**setUserData()****refframe.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

refframe→**start3DCalibration()**
refframe.start3DCalibration()

YRefFrame

Initie le processus de calibration tridimensionnelle des capteurs.

```
function start3DCalibration( )
```

Cette calibration est utilisée à bas niveau pour l'estimation inertielle de position et pour améliorer la précision des mesures d'inclinaison. Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→wait_async()refframe.wait_async()

YRefFrame

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.35. Interface de la fonction Relay

La librairie de programmation Yoctopuce permet simplement de changer l'état du relais. Le changement d'état n'est pas persistant: le relais retournera spontanément à sa position de repos dès que le module est mis hors tension ou redémarré. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Pour les modules dotés de deux sorties par relais (relai inverseur), les deux sorties sont appelées A et B, la sortie A correspondant à la position de repos (hors tension) et la sortie B correspondant à l'état actif. Si vous préféreriez l'état par défaut opposé, vous pouvez simplement changer vos fils sur le bornier.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_relay.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YRelay = yoctolib.YRelay;
php	require_once('yocto_relay.php');
c++	#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.

3. Reference

relay→get_hardwareId()

Retourne l'identifiant matériel unique du relais au format SERIAL . FUNCTIONID.

relay→get_logicalName()

Retourne le nom logique du relais.

relay→get_maxTimeOnStateA()

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

relay→get_maxTimeOnStateB()

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

relay→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

relay→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

relay→get_output()

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

relay→get_pulseTimer()

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

relay→get_state()

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

relay→get_stateAtPowerOn()

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

relay→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

relay→isOnline()

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

relay→isOnline_async(callback, context)

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

relay→load(msValidity)

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

relay→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

relay→nextRelay()

Continue l'énumération des relais commencée à l'aide de yFirstRelay().

relay→pulse(ms_duration)

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

relay→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

relay→set_logicalName(newval)

Modifie le nom logique du relais.

relay→set_maxTimeOnStateA(newval)

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

relay→set_maxTimeOnStateB(newval)

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

relay→**set_output(newval)**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

relay→**set_state(newval)**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

relay→**set_stateAtPowerOn(newval)**

Pré-programme l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

relay→**set_userdata(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

relay→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRelay.FindRelay() yFindRelay()yFindRelay()

YRelay

Permet de retrouver un relais d'après un identifiant donné.

```
function yFindRelay( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le relais soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRelay.isOnline()` pour tester si le relais est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le relais sans ambiguïté

Retourne :

un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

YRelay.FirstRelay()
yFirstRelay()**YRelay**

Commence l'énumération des relais accessibles par la librairie.

```
function yFirstRelay( )
```

Utiliser la fonction `YRelay.nextRelay()` pour itérer sur les autres relais.

Retourne :

un pointeur sur un objet `YRelay`, correspondant au premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

relay→**delayedPulse()****relay.delayedPulse()**

YRelay

Pré-programme une impulsion

```
function delayedPulse( ms_delay, ms_duration)
```

Paramètres :

ms_delay delai d'attente avant l'impulsion, en millisecondes

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**describe()****relay.describe()****YRelay**

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le relais (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

relay→**get_advertisedValue()**

YRelay

relay→**advertisedValue()****relay.get_advertisedValue()**

Retourne la valeur courante du relais (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du relais (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

relay→**get_countdown()****YRelay****relay**→**countdown()****relay.get_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

```
function get_countdown( )
```

Si aucune impulsion n'est programmée, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

relay→**get_errorMessage()**

YRelay

relay→**errorMessage()****relay.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du relais.

relay→**get_errorType()****YRelay****relay**→**errorType()****relay.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du relais.

relay→**get_friendlyName()**

YRelay

relay→**friendlyName()****relay.get_friendlyName()**

Retourne un identifiant global du relais au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du relais si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du relais (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le relais en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

relay→**get_functionDescriptor()****YRelay****relay**→**functionDescriptor()****relay.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

relay→**get_functionId()**

YRelay

relay→**functionId()****relay.get_functionId()**

Retourne l'identifiant matériel du relais, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le relais (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

relay→**get_hardwareId()****YRelay****relay**→**hardwareId()****relay.get_hardwareId()**

Retourne l'identifiant matériel unique du relais au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du relais (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le relais (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

relay→**get_logicalName()**

YRelay

relay→**logicalName()****relay.get_logicalName()**

Retourne le nom logique du relais.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du relais. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

relay→**get_maxTimeOnStateA()****YRelay****relay**→**maxTimeOnStateA()****relay.get_maxTimeOnStateA()**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
function get_maxTimeOnStateA( )
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEA_INVALID`.

relay→**get_maxTimeOnStateB()**

YRelay

relay→**maxTimeOnStateB()**

relay.get_maxTimeOnStateB()

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
function get_maxTimeOnStateB( )
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEB_INVALID`.

relay→**get_module()**
relay→**module()****relay.get_module()**

YRelay

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

relay→**get_module_async()**

YRelay

relay→**module_async()****relay.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

relay→**get_output()****YRelay****relay**→**output()****relay.get_output()**

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
function get_output( )
```

Retourne :

soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUT_INVALID`.

relay→**get_pulseTimer()**

YRelay

relay→**pulseTimer()****relay.get_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
function get_pulseTimer( )
```

Si aucune impulsion n'est en cours, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne `Y_PULSETIMER_INVALID`.

relay→**get_state()****YRelay****relay**→**state()****relay.get_state()**

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

```
function get_state( )
```

Retourne :

soit Y_STATE_A, soit Y_STATE_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y_STATE_INVALID.

relay→**get_stateAtPowerOn()**

YRelay

relay→**stateAtPowerOn()****relay.get_stateAtPowerOn()**

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function get_stateAtPowerOn( )
```

Retourne :

une valeur parmi Y_STATEATPOWERON_UNCHANGED, Y_STATEATPOWERON_A et Y_STATEATPOWERON_B représentant l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y_STATEATPOWERON_INVALID.

relay→**get_userData()****YRelay****relay**→**userData()****relay.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es 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.

fonction **isOnline()** ()

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le relais est joignable, `false` sinon

relay→**isOnline_async()****relay.isOnline_async()****YRelay**

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

relay→**load()****relay.load()****YRelay**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**load_async()****relay.load_async()****YRelay**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

relay→**nextRelay()**relay.nextRelay()

YRelay

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

```
function nextRelay( )
```

Retourne :

un pointeur sur un objet `YRelay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

relay→**pulse()****relay.pulse()****YRelay**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
function pulse( ms_duration)
```

Paramètres :

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**registerValueCallback()**
relay.registerValueCallback()**YRelay**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

relay→**set_logicalName()****YRelay****relay**→**setLogicalName()****relay.set_logicalName()**

Modifie le nom logique du relais.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du relais.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_maxTimeOnStateA()**

YRelay

relay→**setMaxTimeOnStateA()**

relay.set_maxTimeOnStateA()

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
function set_maxTimeOnStateA( newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_maxTimeOnStateB()**

YRelay

relay→**setMaxTimeOnStateB()****relay.set_maxTimeOnStateB()**

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
function set_maxTimeOnStateB( newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_output()**

YRelay

relay→**setOutput()****relay.set_output()**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
function set_output( newval)
```

Paramètres :

newval soit Y_OUTPUT_OFF, soit Y_OUTPUT_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_state()****YRelay****relay**→**setState()****relay.set_state()**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
function set_state( newval)
```

Paramètres :

newval soit Y_STATE_A, soit Y_STATE_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_stateAtPowerOn()**

YRelay

relay→**setStateAtPowerOn()**

relay.set_stateAtPowerOn()

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function set_stateAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_userdata()****YRelay****relay**→**setUserData()****relay.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

relay→wait_async()relay.wait_async()

YRelay

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.36. Interface des fonctions de type senseur

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes 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 retournée par le capteur (sans arrondi ni calibration).

sensor→get_currentValue()

Retourne la valeur actuelle de la mesure.

sensor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

sensor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

sensor→get_friendlyName()

Retourne un identifiant global du senseur au format `NOM_MODULE . NOM_FONCTION`.

sensor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

sensor→get_functionId()

Retourne l'identifiant matériel du senseur, sans référence au module.

sensor→get_hardwareId()

<code>sensor</code>	Retourne l'identifiant matériel unique du capteur au format <code>SERIAL.FUNCTIONID</code> .
<code>sensor</code> → <code>get_highestValue()</code>	Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.
<code>sensor</code> → <code>get_logFrequency()</code>	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.
<code>sensor</code> → <code>get_logicalName()</code>	Retourne le nom logique du capteur.
<code>sensor</code> → <code>get_lowestValue()</code>	Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.
<code>sensor</code> → <code>get_module()</code>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<code>sensor</code> → <code>get_module_async(callback, context)</code>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<code>sensor</code> → <code>get_recordedData(startTime, endTime)</code>	Retourne un objet <code>DataSet</code> représentant des mesures de ce capteur précédemment enregistrées à l'aide du <code>DataLogger</code> , pour l'intervalle de temps spécifié.
<code>sensor</code> → <code>get_reportFrequency()</code>	Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.
<code>sensor</code> → <code>get_resolution()</code>	Retourne la résolution des valeurs mesurées.
<code>sensor</code> → <code>get_unit()</code>	Retourne l'unité dans laquelle la mesure est exprimée.
<code>sensor</code> → <code>get_userData()</code>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<code>sensor</code> → <code>isOnline()</code>	Vérifie si le module hébergeant le capteur est joignable, sans déclencher d'erreur.
<code>sensor</code> → <code>isOnline_async(callback, context)</code>	Vérifie si le module hébergeant le capteur est joignable, sans déclencher d'erreur.
<code>sensor</code> → <code>load(msValidity)</code>	Met en cache les valeurs courantes du capteur, avec une durée de validité spécifiée.
<code>sensor</code> → <code>loadCalibrationPoints(rawValues, refValues)</code>	Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode <code>calibrateFromPoints</code> .
<code>sensor</code> → <code>load_async(msValidity, callback, context)</code>	Met en cache les valeurs courantes du capteur, avec une durée de validité spécifiée.
<code>sensor</code> → <code>nextSensor()</code>	Continue l'énumération des capteurs commencée à l'aide de <code>yFirstSensor()</code> .
<code>sensor</code> → <code>registerTimedReportCallback(callback)</code>	Enregistre la fonction de callback qui est appelée à chaque notification périodique.
<code>sensor</code> → <code>registerValueCallback(callback)</code>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<code>sensor</code> → <code>set_highestValue(newval)</code>	Modifie la mémoire de valeur maximale observée.
<code>sensor</code> → <code>set_logFrequency(newval)</code>	

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

sensor→**set_logicalName(newval)**

Modifie le nom logique du capteur.

sensor→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

sensor→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

sensor→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

sensor→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

sensor→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YSensor.FindSensor() yFindSensor()yFindSensor()

YSensor

Permet de retrouver un senseur d'après un identifiant donné.

```
function yFindSensor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le senseur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSensor.isOnline()` pour tester si le senseur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le senseur sans ambiguïté

Retourne :

un objet de classe `YSensor` qui permet ensuite de contrôler le senseur.

**YSensor.FirstSensor()
yFirstSensor()yFirstSensor()**

YSensor

Commence l'énumération des senseurs accessibles par la librairie.

```
function yFirstSensor( )
```

Utiliser la fonction `YSensor.nextSensor()` pour itérer sur les autres senseurs.

Retourne :

un pointeur sur un objet `YSensor`, correspondant au premier senseur accessible en ligne, ou `null` si il n'y a pas de senseurs disponibles.

sensor→**calibrateFromPoints()**
sensor.calibrateFromPoints()**YSensor**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**describe()****sensor.describe()****YSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le senseur (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

sensor→**get_advertisedValue()**

YSensor

sensor→**advertisedValue()**

sensor.get_advertisedValue()

Retourne la valeur courante du senseur (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du senseur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

sensor→**get_currentRawValue()****YSensor****sensor**→**currentRawValue()****sensor.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

sensor→**get_currentValue()**

YSensor

sensor→**currentValue()****sensor.get_currentValue()**

Retourne la valeur actuelle de la mesure.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de la mesure

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

sensor→**get_errorMessage()****YSensor****sensor**→**errorMessage()****sensor.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la bibliothèque 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.

sensor→**get_errorType()**

YSensor

sensor→**errorType()****sensor.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

sensor→**get_friendlyName()****YSensor****sensor**→**friendlyName()****sensor.get_friendlyName()**

Retourne un identifiant global du senseur au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du senseur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du senseur (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le senseur en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

sensor→**get_functionDescriptor()**

YSensor

sensor→**functionDescriptor()**

sensor.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

sensor→**get_functionId()****YSensor****sensor**→**functionId()****sensor.get_functionId()**

Retourne l'identifiant matériel du senseur, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le senseur (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

sensor→**get_hardwareId()**

YSensor

sensor→**hardwareId()****sensor.get_hardwareId()**

Retourne l'identifiant matériel unique du senseur au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du senseur (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le senseur (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

sensor→**get_highestValue()****YSensor****sensor**→**highestValue()****sensor.get_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

sensor→**get_logFrequency()**

YSensor

sensor→**logFrequency()****sensor.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

sensor→**get_logicalName()****YSensor****sensor**→**logicalName()****sensor.get_logicalName()**

Retourne le nom logique du senseur.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du senseur. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

sensor→**get_lowestValue()**

YSensor

sensor→**lowestValue()****sensor.get_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

sensor→**get_module()****YSensor****sensor**→**module()****sensor.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

sensor→**get_module_async()**

YSensor

sensor→**module_async()****sensor.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

sensor→**get_recordedData()****YSensor****sensor**→**recordedData()****sensor.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez 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()**

YSensor

sensor→**reportFrequency()**

sensor.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

sensor→**get_resolution()****YSensor****sensor**→**resolution()****sensor.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

sensor→**get_unit()**

YSensor

sensor→**unit()****sensor.get_unit()**

Retourne l'unité dans laquelle la mesure est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

sensor→**get_userData()****YSensor****sensor**→**userData()****sensor.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

sensor→**isOnline()****sensor.isOnline()**

YSensor

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du senseur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le senseur est joignable, `false` sinon

sensor→**isOnline_async()****sensor.isOnline_async()****YSensor**

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du senseur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

sensor→**load()****sensor.load()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**loadCalibrationPoints()**
sensor.loadCalibrationPoints()**YSensor**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**load_async()****sensor.load_async()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

sensor→**nextSensor()****sensor.nextSensor()****YSensor**

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

```
function nextSensor( )
```

Retourne :

un pointeur sur un objet `YSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

**sensor→registerTimedReportCallback()
sensor.registerTimedReportCallback()**

YSensor

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

sensor→**set_highestValue()**

YSensor

sensor→**setHighestValue()****sensor.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_logFrequency()**
sensor→**setLogFrequency()**
sensor.set_logFrequency()

YSensor

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_logicalName()**

YSensor

sensor→**setLogicalName()****sensor.set_logicalName()**

Modifie le nom logique du senseur.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du senseur.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_lowestValue()****YSensor****sensor**→**setLowestValue()****sensor.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_reportFrequency()**

YSensor

sensor→**setReportFrequency()**

sensor.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_resolution()****YSensor****sensor**→**setResolution()****sensor.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_userdata()**

YSensor

sensor→**setUserData()****sensor.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( 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

sensor→**wait_async()****sensor.wait_async()****YSensor**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.37. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'intervalle de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_servo.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YServo = yoctolib.YServo;</code>
php	<code>require_once('yocto_servo.php');</code>
c++	<code>#include "yocto_servo.h"</code>
m	<code>#import "yocto_servo.h"</code>
pas	<code>uses yocto_servo;</code>
vb	<code>yocto_servo.vb</code>
cs	<code>yocto_servo.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YServo;</code>
py	<code>from yocto_servo import *</code>

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

Attendez que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelez le callback passé en paramètre.

YServo.FindServo() yFindServo()yFindServo()

YServo

Permet de retrouver un servo d'après un identifiant donné.

```
function yFindServo( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le servo soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le servo sans ambiguïté

Retourne :

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

YServo.FirstServo() yFirstServo()yFirstServo()

YServo

Commence l'énumération des servo accessibles par la librairie.

```
function yFirstServo( )
```

Utiliser la fonction `YServo.nextServo()` pour itérer sur les autres servo.

Retourne :

un pointeur sur un objet `YServo`, correspondant au premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

servo→describe()**servo.describe()****YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le servo (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

servo→**get_advertisedValue()**

YServo

servo→**advertisedValue()****servo.get_advertisedValue()**

Retourne la valeur courante du servo (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du servo (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

servo→**get_enabled()****YServo****servo**→**enabled()****servo.get_enabled()**

Retourne l'état de fonctionnement du \$FUNCTION\$.

```
function get_enabled( )
```

Retourne :

soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon l'état de fonctionnement du \$FUNCTION\$

En cas d'erreur, déclenche une exception ou retourne Y_ENABLED_INVALID.

servo→**get_enabledAtPowerOn()**

YServo

servo→**enabledAtPowerOn()**

servo.get_enabledAtPowerOn()

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

```
function get_enabledAtPowerOn( )
```

Retourne :

soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLEDATPOWERON_INVALID`.

servo→**get_errorMessage()****YServo****servo**→**errorMessage()****servo.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du servo.

servo→**get_errorType()**

YServo

servo→**errorType()****servo.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du servo.

servo→**get_friendlyName()****YServo****servo**→**friendlyName()****servo.get_friendlyName()**

Retourne un identifiant global du servo au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du servo si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du servo (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le servo en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

servo→**get_functionDescriptor()**

YServo

servo→**functionDescriptor()**

servo.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

servo→**get_functionId()****YServo****servo**→**functionId()****servo.get_functionId()**

Retourne l'identifiant matériel du servo, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le servo (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

servo→**get_hardwareId()**

YServo

servo→**hardwareId()****servo.get_hardwareId()**

Retourne l'identifiant matériel unique du servo au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du servo (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le servo (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

servo→**get_logicalName()****YServo****servo**→**logicalName()****servo.get_logicalName()**

Retourne le nom logique du servo.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du servo. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

servo→**get_module()**

YServo

servo→**module()****servo.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

servo→**get_module_async()****YServo****servo**→**module_async()****servo.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

servo→**get_neutral()**

YServo

servo→**neutral()****servo.get_neutral()**

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

```
function get_neutral( )
```

Retourne :

un entier représentant la durée en microsecondes de l'impulsion correspondant au neutre du servo

En cas d'erreur, déclenche une exception ou retourne `Y_NEUTRAL_INVALID`.

servo→**get_position()**
servo→**position()****servo.get_position()**

YServo

Retourne la position courante du servo.

```
function get_position( )
```

Retourne :

un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne `Y_POSITION_INVALID`.

servo→**get_positionAtPowerOn()**

YServo

servo→**positionAtPowerOn()**

servo.get_positionAtPowerOn()

Retourne la position du servo au démarrage du module.

```
function get_positionAtPowerOn( )
```

Retourne :

un entier représentant la position du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_POSITIONATPOWERON_INVALID`.

servo→**get_range()**
servo→**range()****servo.get_range()**

YServo

Retourne la plage d'utilisation du servo.

```
function get_range( )
```

Retourne :

un entier représentant la plage d'utilisation du servo

En cas d'erreur, déclenche une exception ou retourne `Y_RANGE_INVALID`.

servo→**get_userData()**

YServo

servo→**userData()****servo.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

servo→isOnline()**servo.isOnline()****YServo**

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du servo sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le servo est joignable, `false` sinon

servo→**isOnline_async()****servo.isOnline_async()**

YServo

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du servo sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

servo→load()**servo.load()****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**load_async()****servo.load_async()****YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

servo→move()**servo.move()****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

```
function move( target, ms_duration)
```

Paramètres :

target nouvelle position à la fin du mouvement
ms_duration durée totale du mouvement, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**nextServo()****servo.nextServo()**

YServo

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

```
function nextServo( )
```

Retourne :

un pointeur sur un objet `YServo` accessible en ligne, ou `null` lorsque l'énumération est terminée.

servo→**registerValueCallback()**
servo.registerValueCallback()

YServo

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

servo→**set_enabled()**

YServo

servo→**setEnabled()****servo.set_enabled()**

Démarre ou arrête le \$FUNCTION\$.

```
function set_enabled( newval)
```

Paramètres :

newval soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_enabledAtPowerOn()**
servo→**setEnabledAtPowerOn()**
servo.set_enabledAtPowerOn()

YServo

Configure l'état du générateur de signal de commande du servo au démarrage du module.

```
function set_enabledAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_logicalName()**

YServo

servo→**setLogicalName()****servo.set_logicalName()**

Modifie le nom logique du servo.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du servo.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_neutral()****YServo****servo**→**setNeutral()****servo.set_neutral()**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
function set_neutral( newval)
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

Paramètres :

newval un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_position()**

YServo

servo→**setPosition()****servo.set_position()**

Modifie immédiatement la consigne de position du servo.

```
function set_position( newval)
```

Paramètres :

newval un entier représentant immédiatement la consigne de position du servo

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_positionAtPowerOn()**
servo→**setPositionAtPowerOn()**
servo.set_positionAtPowerOn()

YServo

Configure la position du servo au démarrage du module.

```
function set_positionAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_range()**

YServo

servo→**setRange()****servo.set_range()**

Modifie la plage d'utilisation du servo, en pourcents.

```
function set_range( newval)
```

La valeur 100% correspond à un signal de commande standard, variant de 1 [ms] à 2 [ms]. Pour les servos supportent une plage double, de 0.5 [ms] à 2.5 [ms], vous pouvez utiliser une valeur allant jusqu'à 200%. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

Paramètres :

newval un entier représentant la plage d'utilisation du servo, en pourcents

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_userdata()****YServo****servo**→**setUserData()****servo.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

servo→wait_async()servo.wait_async()

YServo

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.38. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les 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');
c++	#include "yocto_temperature.h"
m	#import "yocto_temperature.h"
pas	uses yocto_temperature;
vb	yocto_temperature.vb
cs	yocto_temperature.cs
java	import com.yoctopuce.YoctoAPI.YTemperature;
py	from yocto_temperature import *

Fonction globales

yFindTemperature(func)

Permet de retrouver un capteur de température d'après un identifiant donné.

yFirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

Méthodes des objets YTemperature

temperature→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

temperature→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

temperature→get_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

temperature→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

temperature→get_currentValue()

Retourne la valeur actuelle de la température.

temperature→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

temperature→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

temperature→get_friendlyName()

Retourne un identifiant global du capteur de température au format `NOM_MODULE . NOM_FONCTION`.

temperature→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

temperature→get_functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

temperature→get_hardwareId()

3. Reference

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.

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(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

temperature→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YTemperature.FindTemperature() yFindTemperature()yFindTemperature()

YTemperature

Permet de retrouver un capteur de température d'après un identifiant donné.

```
function yFindTemperature( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de température soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de température sans ambiguïté

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

**YTemperature.FirstTemperature()
yFirstTemperature()yFirstTemperature()**

YTemperature

Commence l'énumération des capteurs de température accessibles par la librairie.

```
function yFirstTemperature( )
```

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Retourne :

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

temperature→calibrateFromPoints() temperature.calibrateFromPoints()

YTemperature

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**describe()****temperature.describe()****YTemperature**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function **describe**()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de température (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.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( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

temperature→**get_currentRawValue()**

YTemperature

temperature→**currentRawValue()**

temperature.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

temperature→**get_currentValue()**

YTemperature

temperature→**currentValue()**

temperature.get_currentValue()

Retourne la valeur actuelle de la température.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de la température

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

temperature→**get_errorMessage()****YTemperature****temperature**→**errorMessage()****temperature.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

temperature→**get_errorType()**
temperature→**errorType()**
temperature.get_errorType()

YTemperature

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

temperature→**get_friendlyName()****YTemperature****temperature**→**friendlyName()****temperature.get_friendlyName()**

Retourne un identifiant global du capteur de température au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de température si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de température (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de température en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

temperature→**get_functionDescriptor()**

YTemperature

temperature→**functionDescriptor()**

temperature.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

temperature→**get_functionId()****YTemperature****temperature**→**functionId()****temperature.get_functionId()**

Retourne l'identifiant matériel du capteur de température, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

temperature→**get_hardwareId()**
temperature→**hardwareId()**
temperature.get_hardwareId()

YTemperature

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

temperature→**get_highestValue()****YTemperature****temperature**→**highestValue()****temperature.get_highestValue()**

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

temperature→**get_logFrequency()**

YTemperature

temperature→**logFrequency()**

temperature.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

temperature→**get_logicalName()****YTemperature****temperature**→**logicalName()****temperature.get_logicalName()**

Retourne le nom logique du capteur de température.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de température. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

temperature→**get_lowestValue()**
temperature→**lowestValue()**
temperature.get_lowestValue()

YTemperature

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

temperature→**get_module()****YTemperature****temperature**→**module()****temperature.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

temperature→get_module_async()

YTemperature

temperature→module_async()

temperature.get_module_async()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

temperature→**get_recordedData()****YTemperature****temperature**→**recordedData()****temperature.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

temperature→**get_reportFrequency()**

YTemperature

temperature→**reportFrequency()**

temperature.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

temperature→**get_resolution()****YTemperature****temperature**→**resolution()****temperature.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

temperature→**get_sensorType()**
temperature→**sensorType()**
temperature.get_sensorType()

YTemperature

Retourne le type de capteur de température utilisé par le module

```
function get_sensorType( )
```

Retourne :

une valeur parmi `Y_SENSORTYPE_DIGITAL`, `Y_SENSORTYPE_TYPE_K`, `Y_SENSORTYPE_TYPE_E`, `Y_SENSORTYPE_TYPE_J`, `Y_SENSORTYPE_TYPE_N`, `Y_SENSORTYPE_TYPE_R`, `Y_SENSORTYPE_TYPE_S`, `Y_SENSORTYPE_TYPE_T`, `Y_SENSORTYPE_PT100_4WIRES`, `Y_SENSORTYPE_PT100_3WIRES` et `Y_SENSORTYPE_PT100_2WIRES` représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORTYPE_INVALID`.

temperature→**get_unit()****YTemperature****temperature**→**unit()****temperature.get_unit()**

Retourne l'unité dans laquelle la température est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

temperature→**get_userData()**

YTemperature

temperature→**userData()****temperature.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

temperature→**isOnline()****temperature.isOnline()****YTemperature**

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de température est joignable, `false` sinon

temperature→isOnline_async() temperature.isOnline_async()

YTemperature

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

temperature→**load()****temperature.load()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**loadCalibrationPoints()**
temperature.loadCalibrationPoints()

YTemperature

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**load_async()**(**temperature.load_async()**)**YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

temperature→**nextTemperature()**
temperature.nextTemperature()

YTemperature

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

```
function nextTemperature()
```

Retourne :

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

temperature→**registerTimedReportCallback()**
temperature.registerTimedReportCallback()

YTemperature

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

temperature→**registerValueCallback()**
temperature.registerValueCallback()

YTemperature

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

temperature→**set_highestValue()****YTemperature****temperature**→**setHighestValue()****temperature.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_logFrequency()**
temperature→**setLogFrequency()**
temperature.set_logFrequency()

YTemperature

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_logicalName()****YTemperature****temperature**→**setLogicalName()****temperature.set_logicalName()**

Modifie le nom logique du capteur de température.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de température.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`temperature`→`set_lowestValue()`

YTemperature

`temperature`→`setLowestValue()`

`temperature.set_lowestValue()`

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_reportFrequency()****YTemperature****temperature**→**setReportFrequency()****temperature.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_resolution()**

YTemperature

temperature→**setResolution()**

temperature.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_sensorType()****YTemperature****temperature**→**setSensorType()****temperature.set_sensorType()**

Change le type de senseur utilisé par le module.

```
function set_sensorType( newval)
```

Cette fonction sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_SENSORTYPE_DIGITAL`, `Y_SENSORTYPE_TYPE_K`, `Y_SENSORTYPE_TYPE_E`, `Y_SENSORTYPE_TYPE_J`, `Y_SENSORTYPE_TYPE_N`, `Y_SENSORTYPE_TYPE_R`, `Y_SENSORTYPE_TYPE_S`, `Y_SENSORTYPE_TYPE_T`, `Y_SENSORTYPE_PT100_4WIRES`, `Y_SENSORTYPE_PT100_3WIRES` et `Y_SENSORTYPE_PT100_2WIRES`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_userData()**

YTemperature

temperature→**setUserData()**

temperature.set_userData()

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
function set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

temperature→**wait_async()****temperature.wait_async()****YTemperature**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.39. Interface de la fonction Tilt

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les 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');
c++	#include "yocto_tilt.h"
m	#import "yocto_tilt.h"
pas	uses yocto_tilt;
vb	yocto_tilt.vb
cs	yocto_tilt.cs
java	import com.yoctopuce.YoctoAPI.YTilt;
py	from yocto_tilt import *

Fonction globales

yFindTilt(func)

Permet de retrouver un inclinomètre d'après un identifiant donné.

yFirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

Méthodes des objets YTilt

tilt→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

tilt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

tilt→get_advertisedValue()

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

tilt→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

tilt→get_currentValue()

Retourne la valeur actuelle de l'inclinaison.

tilt→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

tilt→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

tilt→get_friendlyName()

Retourne un identifiant global de l'inclinomètre au format `NOM_MODULE . NOM_FONCTION`.

tilt→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

tilt→get_functionId()

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

tilt→get_hardwareId()

Retourne l'identifiant matériel unique de l'inclinomètre au format `SERIAL . FUNCTIONID`.

tilt→get_highestValue()

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

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.

tilt→get_logicalName()

Retourne le nom logique de l'inclinomètre.

tilt→get_lowestValue()

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

tilt→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

tilt→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

tilt→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.

tilt→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

tilt→get_resolution()

Retourne la résolution des valeurs mesurées.

tilt→get_unit()

Retourne l'unité dans laquelle l'inclinaison est exprimée.

tilt→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

tilt→isOnline()

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

tilt→isOnline_async(callback, context)

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

tilt→load(msValidity)

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

tilt→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

tilt→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

tilt→nextTilt()

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

tilt→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

tilt→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

tilt→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

tilt→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

3. Reference

tilt→**set_logicalName(newval)**

Modifie le nom logique de l'inclinomètre.

tilt→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

tilt→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

tilt→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

tilt→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

tilt→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YTilt.FindTilt() yFindTilt()yFindTilt()

YTilt

Permet de retrouver un inclinomètre d'après un identifiant donné.

```
function yFindTilt( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'inclinomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTilt.isOnline()` pour tester si l'inclinomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'inclinomètre sans ambiguïté

Retourne :

un objet de classe `YTilt` qui permet ensuite de contrôler l'inclinomètre.

YTilt.FirstTilt() yFirstTilt(yFirstTilt())

YTilt

Commence l'énumération des inclinomètres accessibles par la librairie.

```
function yFirstTilt( )
```

Utiliser la fonction `YTilt.nextTilt()` pour itérer sur les autres inclinomètres.

Retourne :

un pointeur sur un objet `YTilt`, correspondant au premier inclinomètre accessible en ligne, ou `null` si il n'y a pas de inclinomètres disponibles.

tilt→**calibrateFromPoints()****tilt.calibrateFromPoints()****YTilt**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**describe()****tilt.describe()****YTilt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'inclinomètre (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

tilt→**get_advertisedValue()****YTilt****tilt**→**advertisedValue()****tilt.get_advertisedValue()**

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

tilt→**get_currentRawValue()**

YTilt

tilt→**currentRawValue()****tilt.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

tilt→**get_currentValue()****YTilt****tilt**→**currentValue()****tilt.get_currentValue()**

Retourne la valeur actuelle de l'inclinaison.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de l'inclinaison

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

tilt→**get_errorMessage()**

YTilt

tilt→**errorMessage()****tilt.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

tilt→**get_errorType()****YTilt****tilt**→**errorType()****tilt.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

tilt→**get_friendlyName()****tilt**→**friendlyName()****tilt.get_friendlyName()**

Retourne un identifiant global de l'inclinomètre au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'inclinomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'inclinomètre (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'inclinomètre en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

tilt→**get_functionDescriptor()****YTilt****tilt**→**functionDescriptor()****tilt.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur 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( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

tilt→**get_hardwareId()****YTilt****tilt**→**hardwareId()****tilt.get_hardwareId()**

Retourne l'identifiant matériel unique de l'inclinomètre au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'inclinomètre (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'inclinomètre (ex: `RELAYLO1-123456.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( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

tilt→**get_logFrequency()****YTilt****tilt**→**logFrequency()****tilt.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

tilt→**get_logicalName()**

YTilt

tilt→**logicalName()****tilt.get_logicalName()**

Retourne le nom logique de l'inclinomètre.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'inclinomètre. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

tilt→**get_lowestValue()****YTilt****tilt**→**lowestValue()****tilt.get_lowestValue()**

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

tilt→**get_module()**

YTilt

tilt→**module()****tilt.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

tilt→**get_module_async()**

YTilt

tilt→**module_async()****tilt.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

tilt→**get_recordedData()****YTilt****tilt**→**recordedData()****tilt.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

tilt→**get_reportFrequency()****YTilt****tilt**→**reportFrequency()****tilt.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

tilt→**get_resolution()**

YTilt

tilt→**resolution()****tilt.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

tilt→**get_unit()****YTilt****tilt**→**unit()****tilt.get_unit()**

Retourne l'unité dans laquelle l'inclinaison est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle l'inclinaison est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

tilt→**get_userData()**

YTilt

tilt→**userData()****tilt.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

tilt→**isOnline()****tilt.isOnline()****YTilt**

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'inclinomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'inclinomètre est joignable, `false` sinon

tilt→**isOnline_async()****tilt.isOnline_async()****YTilt**

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'inclinomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

tilt→load()tilt.load()**YTilt**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**loadCalibrationPoints()**
tilt.loadCalibrationPoints()

YTilt

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→load_async()tilt.load_async()

YTilt

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

tilt→**nextTilt()****tilt.nextTilt()**

YTilt

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

```
function nextTilt( )
```

Retourne :

un pointeur sur un objet `YTilt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

tilt→**registerTimedReportCallback()**
tilt.registerTimedReportCallback()

YTilt

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

tilt→**registerValueCallback()**
tilt.registerValueCallback()

YTilt

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

tilt→**set_highestValue()**

YTilt

tilt→**setHighestValue()****tilt.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_logFrequency()**

YTilt

tilt→**setLogFrequency()****tilt.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_logicalName()****YTilt****tilt**→**setLogicalName()****tilt.set_logicalName()**

Modifie le nom logique de l'inclinomètre.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'inclinomètre.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_lowestValue()**

YTilt

tilt→**setLowestValue()****tilt.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_reportFrequency()**

YTilt

tilt→**setReportFrequency()****tilt.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_resolution()**

YTilt

tilt→**setResolution()****tilt.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_userdata()**

YTilt

tilt→**setUserData()****tilt.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

tilt→wait_async()tilt.wait_async()

YTilt

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.40. Interface de la fonction Voc

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les 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');
c++	#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 retournée par le capteur (sans arrondi ni calibration).

voc→get_currentValue()

Retourne la mesure actuelle du taux de VOC estimé.

voc→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→get_friendlyName()

Retourne un identifiant global du capteur de Composés Organiques Volatils au format NOM_MODULE . NOM_FONCTION.

voc→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

voc→get_functionId()

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

3. Reference

`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.

`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.

`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)`

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

voc→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

voc→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

voc→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

voc→set_logicalName(newval)

Modifie le nom logique du capteur de Composés Organiques Volatils.

voc→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

voc→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

voc→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

voc→set_userData(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( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de Composés Organiques Volatils soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoc.isOnline()` pour tester si le capteur de Composés Organiques Volatils est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de Composés Organiques Volatils sans ambiguïté

Retourne :

un objet de classe `YVoc` qui permet ensuite de contrôler le capteur de Composés Organiques Volatils.

**YVoc.FirstVoc()
yFirstVoc()yFirstVoc()**

YVoc

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

```
function yFirstVoc( )
```

Utiliser la fonction `YVoc.nextVoc()` pour itérer sur les autres capteurs de Composés Organiques Volatils.

Retourne :

un pointeur sur un objet `YVoc`, correspondant au premier capteur de Composés Organiques Volatils accessible en ligne, ou `null` si il n'y a pas de capteurs de Composés Organiques Volatils disponibles.

voc→**calibrateFromPoints()****voc.calibrateFromPoints()****YVoc**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→describe()**voc.describe()****YVoc**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

voc→**get_advertisedValue()**

YVoc

voc→**advertisedValue()****voc.get_advertisedValue()**

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

```
function get_advertisedValue() ( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

voc→**get_currentRawValue()****YVoc****voc**→**currentRawValue()****voc.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

voc→**get_currentValue()**

YVoc

voc→**currentValue()****voc.get_currentValue()**

Retourne la mesure actuelle du taux de VOC estimé.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la mesure actuelle du taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

voc→**get_errorMessage()****YVoc****voc**→**errorMessage()****voc.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→**get_errorType()**

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( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→**get_friendlyName()****YVoc****voc**→**friendlyName()****voc.get_friendlyName()**

Retourne un identifiant global du capteur de Composés Organiques Volatils au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de Composés Organiques Volatils si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

voc→**get_functionDescriptor()**

YVoc

voc→**functionDescriptor()**

voc.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

voc→**get_functionId()****YVoc****voc**→**functionId()****voc.get_functionId()**

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

voc→**get_hardwareId()**

YVoc

voc→**hardwareId()****voc.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

voc→**get_highestValue()****YVoc****voc**→**highestValue()****voc.get_highestValue()**

Retourne la valeur maximale observée pour le taux de VOC estimé.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour le taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

voc→**get_logFrequency()**

YVoc

voc→**logFrequency()****voc.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

voc→**get_logicalName()****YVoc****voc**→**logicalName()****voc.get_logicalName()**

Retourne le nom logique du capteur de Composés Organiques Volatils.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

voc→**get_lowestValue()**

YVoc

voc→**lowestValue()****voc.get_lowestValue()**

Retourne la valeur minimale observée pour le taux de VOC estimé.

```
function get_lowestValue() ( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour le taux de VOC estimé

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

voc→**get_module()****YVoc****voc**→**module()****voc.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

voc→**get_module_async()****YVoc****voc**→**module_async()****voc.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

voc→**get_recordedData()****YVoc****voc**→**recordedData()****voc.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

voc→**get_reportFrequency()**

YVoc

voc→**reportFrequency()****voc.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

voc→**get_resolution()****YVoc****voc**→**resolution()****voc.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

voc→**get_unit()**

YVoc

voc→**unit()****voc.get_unit()**

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le taux de VOC estimé est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

voc→**get_userData()****YVoc****voc**→**userData()****voc.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

voc→**isOnline()**(**voc.isOnline()**)

YVoc

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de Composés Organiques Volatils sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de Composés Organiques Volatils est joignable, `false` sinon

voc→**isOnline_async()****voc.isOnline_async()****YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context )
```

Si les valeurs des attributs en cache du capteur de Composés Organiques Volatils sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

voc→**load()**(**voc.load()**)**YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→loadCalibrationPoints()
voc.loadCalibrationPoints()**YVoc**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**load_async(voc.load_async())****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

voc→**nextVoc()****voc.nextVoc()****YVoc**

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

```
function nextVoc( )
```

Retourne :

un pointeur sur un objet `YVoc` accessible en ligne, ou `null` lorsque l'énumération est terminée.

voc→**registerTimedReportCallback()**
voc.registerTimedReportCallback()

YVoc

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas 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( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

voc→**set_highestValue()**

YVoc

voc→**setHighestValue()****voc.set_highestValue()**

Modifie la mémoire de valeur maximale observée pour le taux de VOC estimé.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour le taux de VOC estimé

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_logFrequency()****YVoc****voc**→**setLogFrequency()****voc.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_logicalName()**

YVoc

voc→**setLogicalName()****voc.set_logicalName()**

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_lowestValue()****YVoc****voc**→**setLowestValue()****voc.set_lowestValue()**

Modifie la mémoire de valeur minimale observée pour le taux de VOC estimé.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour le taux de VOC estimé

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_reportFrequency()**

YVoc

voc→**setReportFrequency()**

voc.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_resolution()****YVoc****voc**→**setResolution()****voc.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_userData()**

YVoc

voc→**setUserData()****voc.set_userData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
function set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

voc→**wait_async()****voc.wait_async()****YVoc**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.41. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_voltage.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YVoltage = yoctolib.YVoltage;</code>
php	<code>require_once('yocto_voltage.php');</code>
c++	<code>#include "yocto_voltage.h"</code>
m	<code>#import "yocto_voltage.h"</code>
pas	<code>uses yocto_voltage;</code>
vb	<code>yocto_voltage.vb</code>
cs	<code>yocto_voltage.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YVoltage;</code>
py	<code>from yocto_voltage import *</code>

Fonction globales

yFindVoltage(func)

Permet de retrouver un capteur de tension d'après un identifiant donné.

yFirstVoltage()

Commence l'énumération des capteurs de tension accessibles par la librairie.

Méthodes des objets YVoltage

voltage→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

voltage→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

voltage→get_advertisedValue()

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

voltage→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

voltage→get_currentValue()

Retourne la valeur instantanée de la tension.

voltage→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

voltage→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

voltage→get_friendlyName()

Retourne un identifiant global du capteur de tension au format `NOM_MODULE . NOM_FONCTION`.

voltage→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

voltage→get_functionId()

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

voltage→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL . FUNCTIONID.

voltage→**get_highestValue()**

Retourne la valeur maximale observée pour la tension.

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.

voltage→**get_logicalName()**

Retourne le nom logique du capteur de tension.

voltage→**get_lowestValue()**

Retourne la valeur minimale observée pour la tension.

voltage→**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

voltage→**get_module_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

voltage→**get_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.

voltage→**get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

voltage→**get_resolution()**

Retourne la résolution des valeurs mesurées.

voltage→**get_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

voltage→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

voltage→**isOnline()**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

voltage→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

voltage→**load(msValidity)**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

voltage→**loadCalibrationPoints(rawValues, refValues)**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

voltage→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

voltage→**nextVoltage()**

Continue l'énumération des capteurs de tension commencée à l'aide de yFirstVoltage().

voltage→**registerTimedReportCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

voltage→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

voltage→**set_highestValue(newval)**

Modifie la mémoire de valeur maximale observée pour la tension.

voltage→**set_logFrequency(newval)**

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

voltage→**set_logicalName(newval)**

Modifie le nom logique du capteur de tension.

voltage→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée pour la tension.

voltage→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

voltage→**set_resolution(newval)**

Modifie la résolution des valeurs mesurées.

voltage→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

voltage→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YVoltage.FindVoltage() yFindVoltage()yFindVoltage()

YVoltage

Permet de retrouver un capteur de tension d'après un identifiant donné.

```
function yFindVoltage( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltage.isOnline()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de tension sans ambiguïté

Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

YVoltage.FirstVoltage() yFirstVoltage()yFirstVoltage()

YVoltage

Commence l'énumération des capteurs de tension accessibles par la librairie.

```
function yFirstVoltage( )
```

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

Retourne :

un pointeur sur un objet `YVoltage`, correspondant au premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

voltage→**calibrateFromPoints()**
voltage.calibrateFromPoints()**YVoltage**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
function calibrateFromPoints( rawValues, refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→describe()**voltage.describe()****YVoltage**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

function describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de tension (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

voltage→`get_advertisedValue()`**YVoltage****voltage**→`advertisedValue()`**voltage**.`get_advertisedValue()`

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

voltage→**get_currentRawValue()**

YVoltage

voltage→**currentRawValue()**

voltage.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

```
function get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

voltage→**get_currentValue()****YVoltage****voltage**→**currentValue()****voltage.get_currentValue()**

Retourne la valeur instantanée de la tension.

```
function get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur instantanée de la tension

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

voltage→**get_errorMessage()**

YVoltage

voltage→**errorMessage()****voltage.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

voltage→**get_errorType()****YVoltage****voltage**→**errorType()****voltage.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

voltage→**get_friendlyName()**

YVoltage

voltage→**friendlyName()****voltage.get_friendlyName()**

Retourne un identifiant global du capteur de tension au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de tension (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

voltage→**get_functionDescriptor()****YVoltage****voltage**→**functionDescriptor()****voltage.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

voltage→**get_functionId()**

YVoltage

voltage→**functionId()****voltage.get_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

voltage→**get_hardwareId()****YVoltage****voltage**→**hardwareId()****voltage.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de tension au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de tension (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de tension (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

voltage→**get_highestValue()**

YVoltage

voltage→**highestValue()****voltage.get_highestValue()**

Retourne la valeur maximale observée pour la tension.

```
function get_highestValue( )
```

Retourne :

une valeur numérique représentant la valeur maximale observée pour la tension

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

voltage→**get_logFrequency()****YVoltage****voltage**→**logFrequency()****voltage.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
function get_logFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

voltage→**get_logicalName()**

YVoltage

voltage→**logicalName()****voltage.get_logicalName()**

Retourne le nom logique du capteur de tension.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du capteur de tension. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

voltage→**get_lowestValue()****YVoltage****voltage**→**lowestValue()****voltage.get_lowestValue()**

Retourne la valeur minimale observée pour la tension.

```
function get_lowestValue( )
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la tension

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

voltage→**get_module()**

YVoltage

voltage→**module()****voltage.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

voltage→**get_module_async()****YVoltage****voltage**→**module_async()****voltage.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

voltage→**get_recordedData()**

YVoltage

voltage→**recordedData()****voltage.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
function get_recordedData( startTime, endTime)
```

Veillez 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()****YVoltage****voltage**→**reportFrequency()****voltage.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
function get_reportFrequency( )
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

voltage→**get_resolution()**

YVoltage

voltage→**resolution()****voltage.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
function get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

voltage→**get_unit()****YVoltage****voltage**→**unit()****voltage.get_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
function get_unit( )
```

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

voltage→**get_userdata()**

YVoltage

voltage→**userData()****voltage.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

voltage→**isOnline()****voltage.isOnline()****YVoltage**

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de tension est joignable, `false` sinon

voltage→isOnline_async()voltage.isOnline_async()

YVoltage

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

voltage→**load()****voltage.load()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→loadCalibrationPoints()
voltage.loadCalibrationPoints()

YVoltage

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
function loadCalibrationPoints( rawValues, refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**load_async()****voltage.load_async()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

voltage→**nextVoltage()****voltage.nextVoltage()**

YVoltage

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

```
function nextVoltage( )
```

Retourne :

un pointeur sur un objet `YVoltage` accessible en ligne, ou `null` lorsque l'énumération est terminée.

voltage→**registerTimedReportCallback()**
voltage.registerTimedReportCallback()

YVoltage

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
function registerTimedReportCallback( callback )
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

voltage→**registerValueCallback()**
voltage.registerValueCallback()

YVoltage

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

voltage→**set_highestValue()****YVoltage****voltage**→**setHighestValue()****voltage.set_highestValue()**

Modifie la mémoire de valeur maximale observée pour la tension.

```
function set_highestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée pour la tension

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_logFrequency()**

YVoltage

voltage→**setLogFrequency()**

voltage.set_logFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
function set_logFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_logicalName()****YVoltage****voltage**→**setLogicalName()****voltage.set_logicalName()**

Modifie le nom logique du capteur de tension.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de tension.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_lowestValue()**

YVoltage

voltage→**setLowestValue()****voltage.set_lowestValue()**

Modifie la mémoire de valeur minimale observée pour la tension.

```
function set_lowestValue( newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée pour la tension

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_reportFrequency()****YVoltage****voltage**→**setReportFrequency()****voltage.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
function set_reportFrequency( newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_resolution()**

YVoltage

voltage→**setResolution()****voltage.set_resolution()**

Modifie la résolution des valeurs mesurées.

```
function set_resolution( newval)
```

La résolution correspond à la précision de la représentation numérique des mesures. Changer la résolution ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_userdata()****YVoltage****voltage**→**setUserData()****voltage.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

voltage→wait_async()voltage.wait_async()

YVoltage

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.42. Interface de la fonction Source de tension

La librairie de programmation Yoctopuce permet de commander la tension de sortie du module. Vous pouvez affecter une valeur fixe, ou faire des transitions de voltage.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_vsource.js'></script>
php	require_once('yocto_vsource.php');
cpp	#include "yocto_vsource.h"
m	#import "yocto_vsource.h"
pas	uses yocto_vsource;
vb	yocto_vsource.vb
cs	yocto_vsource.cs
java	import com.yoctopuce.YoctoAPI.YVSource;
py	from yocto_vsource import *

Fonction globales	
yFindVSource(func)	Permet de retrouver une source de tension d'après un identifiant donné.
yFirstVSource()	Commence l'énumération des sources de tension accessibles par la librairie.
Méthodes des objets YVSource	
vsource→describe()	Retourne un court texte décrivant la fonction au format TYPE (NAME) =SERIAL . FUNCTIONID.
vsource→get_advertisedValue()	Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
vsource→get_errorMessage()	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
vsource→get_errorType()	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
vsource→get_extPowerFailure()	Rend TRUE si le voltage de l'alimentation externe est trop bas.
vsource→get_failure()	Indique si le module est en condition d'erreur.
vsource→get_friendlyName()	Retourne un identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.
vsource→get_functionDescriptor()	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
vsource→get_functionId()	Retourne l'identifiant matériel de la fonction, sans référence au module.
vsource→get_hardwareId()	Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.
vsource→get_logicalName()	Retourne le nom logique de la source de tension.
vsource→get_module()	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
vsource→get_module_async(callback, context)	

3. Reference

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`vsource`→`get_overCurrent()`

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

`vsource`→`get_overHeat()`

Rend TRUE si le module est en surchauffe.

`vsource`→`get_overLoad()`

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

`vsource`→`get_regulationFailure()`

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

`vsource`→`get_unit()`

Retourne l'unité dans laquelle la tension est exprimée.

`vsource`→`get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

`vsource`→`get_voltage()`

Retourne la valeur de la commande de tension de sortie en mV

`vsource`→`isOnline()`

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

`vsource`→`isOnline_async(callback, context)`

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

`vsource`→`load(msValidity)`

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

`vsource`→`load_async(msValidity, callback, context)`

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

`vsource`→`nextVSource()`

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

`vsource`→`pulse(voltage, ms_duration)`

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

`vsource`→`registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

`vsource`→`set_logicalName(newval)`

Modifie le nom logique de la source de tension.

`vsource`→`set_userData(data)`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

`vsource`→`set_voltage(newval)`

Règle la tension de sortie du module (en milliVolts).

`vsource`→`voltageMove(target, ms_duration)`

Déclenche une variation constante de la sortie vers une valeur donnée.

`vsource`→`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())

YVSource

Permet de retrouver une source de tension d'après un identifiant donné.

```
function yFindVSource( func)
```

yFindVSource() — YVSource.FindVSource(yFindVSource())

Permet de retrouver une source de tension d'après un identifiant donné.

```
js 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( )
```

yFirstVSource() — YVSource.FirstVSource()yFirstVSource()

Commence l'énumération des sources de tension accessibles par la librairie.

```
js function yFirstVSource( )
php function yFirstVSource( )
cpp YVSource* yFirstVSource( )
m YVSource* yFirstVSource( )
pas function yFirstVSource( ): TYVSource
vb function yFirstVSource( ) As YVSource
cs YVSource FirstVSource( )
java YVSource FirstVSource( )
py def FirstVSource( )
```

Utiliser la fonction `YVSource.nextVSource()` pour itérer sur les autres sources de tension.

Retourne :

un pointeur sur un objet `YVSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de sources de tension disponibles.

vsource→**describe()****vsource.describe()****YVSource**

Retourne un court texte décrivant la fonction au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

vsource→**describe()****vsource.describe()**

Retourne un court texte décrivant la fonction au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

js	function describe ()
php	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	function describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant la fonction (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

vsource→**get_advertisedValue()****YVSource****vsource**→**advertisedValue()****vsource.get_advertisedValue()**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

vsource→**get_advertisedValue()****vsource**→**advertisedValue(vsource.get_advertisedValue())**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

js	function get_advertisedValue ()
php	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	function get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
py	def get_advertisedValue ()
cmd	YVSource target get_advertisedValue

Retourne :

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères)

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

vsource→**get_errorMessage()****YVSource****vsource**→**errorMessage()****vsource.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
function get_errorMessage( )
```

vsource→**get_errorMessage()****vsource**→**errorMessage()****vsource.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

vsourc→**get_errorType()**

YVSource

vsourc→**errorType()****vsourc.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
function get_errorType( )
```

vsourc→**get_errorType()**

vsourc→**errorType()****vsourc.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
js function get_errorType( )
```

```
php function get_errorType( )
```

```
cpp YRETCODE get_errorType( )
```

```
pas function get_errorType( ): YRETCODE
```

```
vb function get_errorType( ) As YRETCODE
```

```
cs YRETCODE get_errorType( )
```

```
java int get_errorType( )
```

```
py def get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

vsource→**get_extPowerFailure()****YVSource****vsource**→**extPowerFailure()****vsource.get_extPowerFailure()**

Rend TRUE si le voltage de l'alimentation externe est trop bas.

```
function get_extPowerFailure( )
```

vsource→**get_extPowerFailure()****vsource**→**extPowerFailure(vsource.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( )
```

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 est possible de savoir de quelle erreur il s'agit en testant `get_overheat`, `get_overcurrent` etc... Lorsqu'une condition d'erreur est rencontrée, la tension de sortie est mise à zéro et ne peut pas être changée tant que la fonction `reset()` n'aura pas été appelée.

Retourne :

soit `Y_FAILURE_FALSE`, soit `Y_FAILURE_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_FAILURE_INVALID`.

vsource→**get_friendlyName()****YVSource****vsource**→**friendlyName()****vsource.get_friendlyName()**

 Retourne un identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

vsource→**get_friendlyName()****vsource**→**friendlyName()****vsource.get_friendlyName()**

 Retourne un identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

<code>js</code>	<code>function get_friendlyName()</code>
<code>php</code>	<code>function get_friendlyName()</code>
<code>cpp</code>	<code>virtual string get_friendlyName()</code>
<code>m</code>	<code>-(NSString*) friendlyName</code>
<code>cs</code>	<code>override string get_friendlyName()</code>
<code>java</code>	<code>String get_friendlyName()</code>

Le chaîne retournée utilise soit les noms logiques du module et de la fonction si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la fonction (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la fonction en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

vsource→**get_functionDescriptor()****vsource**→**functionDescriptor()****vsource.get_vsourceDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
function get_functionDescriptor( )
```

vsource→**get_functionDescriptor()****vsource**→**functionDescriptor()****vsource.get_vsourceDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

js	function get_functionDescriptor ()
php	function get_functionDescriptor ()
cpp	YFUN_DESCR get_functionDescriptor ()
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	def get_functionDescriptor ()

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

vsource→**get_functionId()****YVSource****vsource**→**functionId()****vsource.get_vsourceId()**

Retourne l'identifiant matériel de la fonction, sans référence au module.

```
function get_functionId( )
```

vsource→**get_functionId()****vsource**→**functionId()****vsource.get_vsourceId()**

Retourne l'identifiant matériel de la fonction, sans référence au module.

```
js function get_functionId( )  
php function get_functionId( )  
cpp string get_functionId( )  
m -(NSString*) functionId  
vb function get_functionId( ) As String  
cs string get_functionId( )  
java String get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la fonction (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

vsourc→**get_hardwareId()**

YVSource

vsourc→**hardwareId()****vsourc.get_hardwareId()**

Retourne l'identifiant matériel unique de la fonction au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

vsourc→**get_hardwareId()**

vsourc→**hardwareId()****vsourc.get_hardwareId()**

Retourne l'identifiant matériel unique de la fonction au format SERIAL.FUNCTIONID.

```
js function get_hardwareId( )
```

```
php function get_hardwareId( )
```

```
cpp string get_hardwareId( )
```

```
m -(NSString*) hardwareId
```

```
vb function get_hardwareId( ) As String
```

```
cs string get_hardwareId( )
```

```
java String get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la fonction (ex: RELAYLO1-123456.relay1) En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

vsource→**get_logicalName()****YVSource****vsource**→**logicalName()****vsource.get_logicalName()**

Retourne le nom logique de la source de tension.

```
function get_logicalName( )
```

vsource→**get_logicalName()****vsource**→**logicalName()****vsource.get_logicalName()**

Retourne le nom logique de la source de tension.

js	function get_logicalName ()
php	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
py	def get_logicalName ()
cmd	YVSource target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique de la source de tension

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

vsource→**get_module()****vsource**→**module()****vsource.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

vsource→**get_module()****vsource**→**module()****vsource.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function get_module ()
php	function get_module ()
cpp	<code>YModule *</code> get_module ()
m	-(<code>YModule*</code>) module
pas	function get_module (): <code>TYModule</code>
vb	function get_module () As <code>YModule</code>
cs	<code>YModule</code> get_module ()
java	<code>YModule</code> get_module ()
py	def get_module ()

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

vsource→**get_module_async()****YVSource****vsource**→**module_async()****vsource.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

function **get_module_async**(**callback**, **context**)

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

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( )
```

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.

vsource→**get_overHeat()****YVSource****vsource**→**overHeat()****vsource.get_overHeat()**

Rend TRUE si le module est en surchauffe.

function **get_overHeat**()**vsource**→**get_overHeat()****vsource**→**overHeat()****vsource.get_overHeat()**

Rend TRUE si le module est en surchauffe.

js	function get_overHeat ()
php	function get_overHeat ()
cpp	Y_OVERHEAT_enum get_overHeat ()
m	-(Y_OVERHEAT_enum) overHeat
pas	function get_overHeat (): Integer
vb	function get_overHeat () As Integer
cs	int get_overHeat ()
java	int get_overHeat ()
py	def get_overHeat ()
cmd	YVSource target get_overHeat

Retourne :

soit Y_OVERHEAT_FALSE, soit Y_OVERHEAT_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_OVERHEAT_INVALID.

vsource→**get_overLoad()****vsource**→**overLoad()****vsource.get_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

```
function get_overLoad( )
```

vsource→**get_overLoad()****vsource**→**overLoad()****vsource.get_overLoad()**

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

js	function get_overLoad ()
php	function get_overLoad ()
cpp	Y_OVERLOAD_enum get_overLoad ()
m	-(Y_OVERLOAD_enum) overLoad
pas	function get_overLoad (): Integer
vb	function get_overLoad () As Integer
cs	int get_overLoad ()
java	int get_overLoad ()
py	def get_overLoad ()
cmd	YVSource target get_overLoad

Retourne :

soit Y_OVERLOAD_FALSE, soit Y_OVERLOAD_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_OVERLOAD_INVALID.

vsource→**get_regulationFailure()****YVSource****vsource**→**regulationFailure()****vsource.get_regulationFailure()**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

```
function get_regulationFailure( )
```

vsource→**get_regulationFailure()****vsource**→**regulationFailure()****vsource.get_regulationFailure()**

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

<code>js</code>	function get_regulationFailure ()
<code>php</code>	function get_regulationFailure ()
<code>cpp</code>	Y_REGULATIONFAILURE_enum get_regulationFailure ()
<code>m</code>	-(Y_REGULATIONFAILURE_enum) regulationFailure
<code>pas</code>	function get_regulationFailure (): Integer
<code>vb</code>	function get_regulationFailure () As Integer
<code>cs</code>	int get_regulationFailure ()
<code>java</code>	int get_regulationFailure ()
<code>py</code>	def get_regulationFailure ()
<code>cmd</code>	YVSource target get_regulationFailure

Retourne :

soit Y_REGULATIONFAILURE_FALSE, soit Y_REGULATIONFAILURE_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_REGULATIONFAILURE_INVALID.

vsource→**get_unit()****vsource**→**unit()****vsource.get_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

```
function get_unit( )
```

vsource→**get_unit()****vsource**→**unit()****vsource.get_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

js	function get_unit ()
php	function get_unit ()
cpp	string get_unit ()
m	-(NSString*) unit
pas	function get_unit (): string
vb	function get_unit () As String
cs	string get_unit ()
java	String get_unit ()
py	def get_unit ()
cmd	YVSource target get_unit

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

vsource→**get_userData()****YVSource****vsource**→**userData()****vsource.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

```
function get_userData( )
```

vsource→**get_userData()****vsource**→**userData()****vsource.get_userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

js	function get_userData ()
php	function get_userData ()
cpp	void * get_userData ()
m	-(void*) <code>userData</code>
pas	function get_userData (): Tobject
vb	function get_userData () As Object
cs	object get_userData ()
java	Object get_userData ()
py	def get_userData ()

Cet attribut n'es 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( )
```

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( )
```

vsource→**isOnline()****vsource.isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

js	function isOnline ()
php	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si la fonction est joignable, `false` sinon

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

vsource→**load()****vsource.load()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

```
function load( msValidity)
```

vsource→**load()****vsource.load()**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

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 → **load_async()** **vsource.load_async()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

vsource→**nextVSource()****vsource.nextVSource()****YVSource**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

```
function nextVSource( )
```

vsource→**nextVSource()****vsource.nextVSource()**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

js	function nextVSource ()
php	function nextVSource ()
cpp	YVSource * nextVSource ()
m	-(YVSource*) nextVSource
pas	function nextVSource (): TYVSource
vb	function nextVSource () As YVSource
cs	YVSource nextVSource ()
java	YVSource nextVSource ()
py	def nextVSource ()

Retourne :

un pointeur sur un objet `YVSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

vsource→**pulse()****vsource.pulse()****YVSource**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
function pulse( voltage, ms_duration)
```

vsource→**pulse()****vsource.pulse()**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
js  function pulse( voltage, ms_duration)
php  function pulse( $voltage, $ms_duration)
cpp  int pulse( int voltage, int ms_duration)
m    -(int) pulse : (int) voltage : (int) ms_duration
pas  function pulse( voltage: integer, ms_duration: integer): integer
vb   function pulse( ByVal voltage As Integer,
                   ByVal ms_duration As Integer) As Integer
cs   int pulse( int voltage, int ms_duration)
java int pulse( int voltage, int ms_duration)
py   def pulse( voltage, ms_duration)
cmd  YVSource target pulse voltage ms_duration
```

Paramètres :

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.

```
function registerValueCallback( callback)
```

vsource→**registerValueCallback()****vsource.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback (callback)
php	function registerValueCallback (\$callback)
cpp	void registerValueCallback (YDisplayUpdateCallback callback)
pas	procedure registerValueCallback (callback : TGenericUpdateCallback)
vb	procedure registerValueCallback (ByVal callback As GenericUpdateCallback)
cs	void registerValueCallback (UpdateCallback callback)
java	void registerValueCallback (UpdateCallback callback)
py	def registerValueCallback (callback)
m	-(void) registerValueCallback : (YFunctionUpdateCallback) callback

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

vsource→**set_logicalName()****YVSource****vsource**→**setLogicalName()****vsource.set_logicalName()**

Modifie le nom logique de la source de tension.

```
function set_logicalName( newval)
```

vsource→**set_logicalName()****vsource**→**setLogicalName()****vsource.set_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.

vsource→**set_userData()**

YVSource

vsource→**setUserData()****vsource.set_userData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
function set_userData( data)
```

vsource→**set_userData()****vsource**→**setUserData()****vsource.set_userData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

vsource→**set_voltage()****vsource**→**setVoltage()****vsource.set_voltage()**

Règle la tension de sortie du module (en milliVolts).

```
function set_voltage( newval)
```

vsource→**set_voltage()****vsource**→**setVoltage()****vsource.set_voltage()**

Règle la tension de sortie du module (en milliVolts).

js	function set_voltage (newval)
php	function set_voltage (\$newval)
cpp	int set_voltage (int newval)
m	-(int) setVoltage : (int) newval
pas	function set_voltage (newval : LongInt): integer
vb	function set_voltage (ByVal newval As Integer) As Integer
cs	int set_voltage (int newval)
java	int set_voltage (int newval)
py	def set_voltage (newval)
cmd	YVSource target set_voltage newval

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsource→**voltageMove()**vsource.voltageMove()**YVSource**

Déclenche une variation constante de la sortie vers une valeur donnée.

```
function voltageMove( target, ms_duration)
```

vsource→**voltageMove()**vsource.voltageMove()

Déclenche une variation constante de la sortie vers une valeur donnée.

<code>js</code>	<code>function voltageMove(target, ms_duration)</code>
<code>php</code>	<code>function voltageMove(\$target, \$ms_duration)</code>
<code>cpp</code>	<code>int voltageMove(int target, int ms_duration)</code>
<code>m</code>	<code>-(int) voltageMove : (int) target : (int) ms_duration</code>
<code>pas</code>	<code>function voltageMove(target: integer, ms_duration: integer): integer</code>
<code>vb</code>	<code>function voltageMove(ByVal target As Integer, ByVal ms_duration As Integer) As Integer</code>
<code>cs</code>	<code>int voltageMove(int target, int ms_duration)</code>
<code>java</code>	<code>int voltageMove(int target, int ms_duration)</code>
<code>py</code>	<code>def voltageMove(target, ms_duration)</code>
<code>cmd</code>	<code>YVSource target voltageMove target ms_duration</code>

Paramètres :

target nouvelle valeur de sortie à la fin de la transition, en milliVolts.
ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsource → **wait_async()** **vsource.wait_async()**

YVSource

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

function **wait_async**(**callback**, **context**)

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la VM Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.43. Interface de la fonction WakeUpMonitor

La fonction WakeUpMonitor prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wakeupmonitor.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWakeUpMonitor = yoctolib.YWakeUpMonitor;
php	require_once('yocto_wakeupmonitor.php');
cpp	#include "yocto_wakeupmonitor.h"
m	#import "yocto_wakeupmonitor.h"
pas	uses yocto_wakeupmonitor;
vb	yocto_wakeupmonitor.vb
cs	yocto_wakeupmonitor.cs
java	import com.yoctopuce.YoctoAPI.YWakeUpMonitor;
py	from yocto_wakeupmonitor import *

Fonction globales

yFindWakeUpMonitor(func)

Permet de retrouver un moniteur d'après un identifiant donné.

yFirstWakeUpMonitor()

Commence l'énumération des Moniteurs accessibles par la librairie.

Méthodes des objets YWakeUpMonitor

wakeupmonitor→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

wakeupmonitor→get_advertisedValue()

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

wakeupmonitor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

wakeupmonitor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

wakeupmonitor→get_friendlyName()

Retourne un identifiant global du moniteur au format `NOM_MODULE . NOM_FONCTION`.

wakeupmonitor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

wakeupmonitor→get_functionId()

Retourne l'identifiant matériel du moniteur, sans référence au module.

wakeupmonitor→get_hardwareId()

Retourne l'identifiant matériel unique du moniteur au format `SERIAL . FUNCTIONID`.

wakeupmonitor→get_logicalName()

Retourne le nom logique du moniteur.

wakeupmonitor→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wakeupmonitor→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

3. Reference

wakeupmonitor→**get_nextWakeUp()**

Retourne la prochaine date/heure de réveil agendée (format UNIX)

wakeupmonitor→**get_powerDuration()**

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

wakeupmonitor→**get_sleepCountdown()**

Retourne le temps avant le prochain sommeil.

wakeupmonitor→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

wakeupmonitor→**get_wakeUpReason()**

Renvoie la raison du dernier réveil.

wakeupmonitor→**get_wakeUpState()**

Revoie l'état actuel du moniteur

wakeupmonitor→**isOnline()**

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

wakeupmonitor→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

wakeupmonitor→**load(msValidity)**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

wakeupmonitor→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

wakeupmonitor→**nextWakeUpMonitor()**

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

wakeupmonitor→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wakeupmonitor→**resetSleepCountDown()**

Réinitialise le compteur de mise en sommeil.

wakeupmonitor→**set_logicalName(newval)**

Modifie le nom logique du moniteur.

wakeupmonitor→**set_nextWakeUp(newval)**

Modifie les jours de la semaine où un réveil doit avoir lieu.

wakeupmonitor→**set_powerDuration(newval)**

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

wakeupmonitor→**set_sleepCountdown(newval)**

Modifie le temps avant le prochain sommeil .

wakeupmonitor→**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`.

wakeupmonitor→**sleep(secBeforeSleep)**

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.

wakeupmonitor→**sleepFor(secUntilWakeUp, secBeforeSleep)**

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.

wakeupmonitor→**sleepUntil(wakeUpTime, secBeforeSleep)**

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.

wakeupmonitor→**wait_async(callback, context)**

Attendez que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelez le callback passé en paramètre.

wakeupmonitor → **wakeUp()**

Force un réveil.

YWakeUpMonitor.FindWakeUpMonitor() yFindWakeUpMonitor()yFindWakeUpMonitor()

YWakeUpMonitor

Permet de retrouver un moniteur d'après un identifiant donné.

```
function yFindWakeUpMonitor( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.isOnline()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le moniteur sans ambiguïté

Retourne :

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

**YWakeUpMonitor.FirstWakeUpMonitor()
yFirstWakeUpMonitor()yFirstWakeUpMonitor()**

YWakeUpMonitor

Commence l'énumération des Moniteurs accessibles par la librairie.

```
function yFirstWakeUpMonitor( )
```

Utiliser la fonction `YWakeUpMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

Retourne :

un pointeur sur un objet `YWakeUpMonitor`, correspondant au premier moniteur accessible en ligne, ou `null` si il n'y a pas de Moniteurs disponibles.

**wakeupmonitor→describe()
wakeupmonitor.describe()****YWakeUpMonitor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le moniteur (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

wakeupmonitor→**get_advertisedValue()****YWakeUpMonitor****wakeupmonitor**→**advertisedValue()****wakeupmonitor.get_advertisedValue()**

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

`wakeupmonitor`→`get_errorMessage()`

`YWakeUpMonitor`

`wakeupmonitor`→`errorMessage()`

`wakeupmonitor`.`get_errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

wakeupmonitor→**get_errorType()****YWakeUpMonitor****wakeupmonitor**→**errorType()****wakeupmonitor.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

wakeupmonitor→get_friendlyName()

YWakeUpMonitor

wakeupmonitor→friendlyName()

wakeupmonitor.get_friendlyName()

Retourne un identifiant global du moniteur au format NOM_MODULE .NOM_FONCTION.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du moniteur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du moniteur (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le moniteur en utilisant les noms logiques (ex: MyCustomName.relay1) En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

wakeupmonitor→**get_functionDescriptor()****YWakeUpMonitor****wakeupmonitor**→**functionDescriptor()****wakeupmonitor.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

wakeupmonitor→get_functionId()

YWakeUpMonitor

wakeupmonitor→functionId()

wakeupmonitor.get_functionId()

Retourne l'identifiant matériel du moniteur, sans référence au module.

```
function get_functionId( )
```

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le moniteur (ex: relay1) En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

wakeupmonitor→**get_hardwareId()****YWakeUpMonitor****wakeupmonitor**→**hardwareId()****wakeupmonitor.get_hardwareId()**

Retourne l'identifiant matériel unique du moniteur au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moniteur (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le moniteur (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

wakeupmonitor→get_logicalName()

YWakeUpMonitor

wakeupmonitor→logicalName()

wakeupmonitor.get_logicalName()

Retourne le nom logique du moniteur.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du moniteur. En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

wakeupmonitor→**get_module()****YWakeUpMonitor****wakeupmonitor**→**module()****wakeupmonitor.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

wakeupmonitor→get_module_async()

YWakeUpMonitor

wakeupmonitor→module_async()

wakeupmonitor.get_module_async()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de YModule

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupmonitor→**get_nextWakeUp()**
wakeupmonitor→**nextWakeUp()**
wakeupmonitor.get_nextWakeUp()

YWakeUpMonitor

Retourne la prochaine date/heure de réveil agendée (format UNIX)

```
function get_nextWakeUp( )
```

Retourne :

un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne `Y_NEXTWAKEUP_INVALID`.

wakeupmonitor→get_powerDuration()

YWakeUpMonitor

wakeupmonitor→powerDuration()

wakeupmonitor.get_powerDuration()

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
function get_powerDuration( )
```

Retourne :

un entier représentant le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement

En cas d'erreur, déclenche une exception ou retourne Y_POWERDURATION_INVALID.

wakeupmonitor→get_sleepCountdown()

YWakeUpMonitor

wakeupmonitor→sleepCountdown()

wakeupmonitor.get_sleepCountdown()

Retourne le temps avant le prochain sommeil.

```
function get_sleepCountdown( )
```

Retourne :

un entier représentant le temps avant le prochain sommeil

En cas d'erreur, déclenche une exception ou retourne Y_SLEEPDOWNDOWN_INVALID.

wakeupmonitor→get_userdata()

YWakeUpMonitor

wakeupmonitor→userData()

wakeupmonitor.get_userdata()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

```
function get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

wakeupmonitor→**get_wakeUpReason()****YWakeUpMonitor****wakeupmonitor**→**wakeUpReason()****wakeupmonitor.get_wakeUpReason()**

Renvoie la raison du dernier réveil.

function **get_wakeUpReason()** ()**Retourne :**

```
une valeur parmi Y_WAKEUPREASON_USBPOWER, Y_WAKEUPREASON_EXTPOWER,  
Y_WAKEUPREASON_ENDOFSLEEP,           Y_WAKEUPREASON_EXTSIG1,  
Y_WAKEUPREASON_EXTSIG2,               Y_WAKEUPREASON_EXTSIG3,  
Y_WAKEUPREASON_EXTSIG4,               Y_WAKEUPREASON_SCHEDULE1,  
Y_WAKEUPREASON_SCHEDULE2,            Y_WAKEUPREASON_SCHEDULE3,  
Y_WAKEUPREASON_SCHEDULE4,           Y_WAKEUPREASON_SCHEDULE5    et  
Y_WAKEUPREASON_SCHEDULE6
```

En cas d'erreur, déclenche une exception ou retourne `Y_WAKEUPREASON_INVALID`.

wakeupmonitor→get_wakeUpState()

YWakeUpMonitor

wakeupmonitor→wakeUpState()

wakeupmonitor.get_wakeUpState()

Revoie l'état actuel du moniteur

```
function get_wakeUpState( )
```

Retourne :

soit Y_WAKEUPSTATE_SLEEPING, soit Y_WAKEUPSTATE_AWAKE

En cas d'erreur, déclenche une exception ou retourne Y_WAKEUPSTATE_INVALID.

wakeupmonitor→**isOnline()**wakeupmonitor.isOnline()**YWakeUpMonitor**

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du moniteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le moniteur est joignable, `false` sinon

**wakeupmonitor→isOnline_async()
wakeupmonitor.isOnline_async()**

YWakeUpMonitor

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du moniteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupmonitor→**load()**wakeupmonitor.load()**YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**load_async()**
wakeupmonitor.load_async()**YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupmonitor→**nextWakeUpMonitor()**
wakeupmonitor.nextWakeUpMonitor()

YWakeUpMonitor

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

```
function nextWakeUpMonitor( )
```

Retourne :

un pointeur sur un objet `YWakeUpMonitor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

wakeupmonitor→registerValueCallback() wakeupmonitor.registerValueCallback()

YWakeUpMonitor

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wakeupmonitor→**resetSleepCountDown()**
wakeupmonitor.resetSleepCountDown()

YWakeUpMonitor

Réinitialise le compteur de mise en sommeil.

```
function resetSleepCountDown( )
```

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set_logicalName()**
wakeupmonitor→**setLogicalName()**
wakeupmonitor.set_logicalName()

YWakeUpMonitor

Modifie le nom logique du moniteur.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du moniteur.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set_nextWakeUp()****YWakeUpMonitor****wakeupmonitor**→**setNextWakeUp()****wakeupmonitor.set_nextWakeUp()**

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
function set_nextWakeUp( newval)
```

Paramètres :

newval un entier représentant les jours de la semaine où un réveil doit avoir lieu

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_powerDuration()

YWakeUpMonitor

wakeupmonitor→setPowerDuration()

wakeupmonitor.set_powerDuration()

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
function set_powerDuration( newval)
```

Paramètres :

newval un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_sleepCountdown()

YWakeUpMonitor

wakeupmonitor→setSleepCountdown()

wakeupmonitor.set_sleepCountdown()

Modifie le temps avant le prochain sommeil .

```
function set_sleepCountdown( newval)
```

Paramètres :

newval un entier représentant le temps avant le prochain sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_userdata()

YWakeUpMonitor

wakeupmonitor→setUserData()

wakeupmonitor.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
function set_userdata( data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

wakeupmonitor → **sleep()**wakeupmonitor.sleep()**YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
function sleep( secBeforeSleep)
```

Paramètres :

secBeforeSleep nombre de seconde avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**sleepFor()**
wakeupmonitor.sleepFor()

YWakeUpMonitor

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
function sleepFor( secUntilWakeUp, secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à `resetSleepCountDown`.

Paramètres :

secUntilWakeUp durée de la mise en sommeil, en secondes

secBeforeSleep nombre de secondes avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**sleepUntil()**
wakeupmonitor.sleepUntil()**YWakeUpMonitor**

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
function sleepUntil( wakeUpTime, secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à `resetSleepCountDown`.

Paramètres :

wakeUpTime date/heure du réveil (format UNIX)
secBeforeSleep nombre de secondes avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**wait_async()**
wakeupmonitor.wait_async()

YWakeUpMonitor

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

wakeupmonitor→**wakeUp()**wakeupmonitor.wakeUp()**YWakeUpMonitor**

Force un réveil.

```
function wakeUp( )
```

3.44. Interface de la fonction WakeUpSchedule

La fonction WakeUpSchedule implémente une condition de réveil. Le réveil est spécifiée par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wakeupschedule.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWakeUpSchedule = yoctolib.YWakeUpSchedule;
php	require_once('yocto_wakeupschedule.php');
c++	#include "yocto_wakeupschedule.h"
m	#import "yocto_wakeupschedule.h"
pas	uses yocto_wakeupschedule;
vb	yocto_wakeupschedule.vb
cs	yocto_wakeupschedule.cs
java	import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py	from yocto_wakeupschedule import *

Fonction globales

yFindWakeUpSchedule(func)

Permet de retrouver un réveil agendé d'après un identifiant donné.

yFirstWakeUpSchedule()

Commence l'énumération des réveils agendés accessibles par la librairie.

Méthodes des objets YWakeUpSchedule

wakeupschedule→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

wakeupschedule→get_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

wakeupschedule→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

wakeupschedule→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

wakeupschedule→get_friendlyName()

Retourne un identifiant global du réveil agendé au format `NOM_MODULE . NOM_FONCTION`.

wakeupschedule→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

wakeupschedule→get_functionId()

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

wakeupschedule→get_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format `SERIAL . FUNCTIONID`.

wakeupschedule→get_hours()

Retourne les heures où le réveil est actif..

wakeupschedule→get_logicalName()

Retourne le nom logique du réveil agendé.

wakeupschedule→get_minutes()

Retourne toutes les minutes de chaque heure où le réveil est actif.

wakeupschedule→get_minutesA()

Retourne les minutes de l'intervall 00-29 de chaque heures où le réveil est actif.

wakeupschedule→get_minutesB()

Retourne les minutes de l'intervall 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_nextOccurence()

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'intervall 00-29 où un réveil doit avoir lieu

wakeupschedule→set_minutesB(newval)

Modifie les minutes de l'intervall 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)

3. Reference

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

wakeupschedule → **set_weekDays**(**newval**, **newval**)

Modifie les jours de la semaine où un réveil doit avoir lieu

wakeupschedule → **wait_async**(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWakeUpSchedule.FindWakeUpSchedule() yFindWakeUpSchedule()yFindWakeUpSchedule()

YWakeUpSchedule

Permet de retrouver un réveil agendé d'après un identifiant donné.

```
function yFindWakeUpSchedule( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le réveil agendé soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpSchedule.isOnline()` pour tester si le réveil agendé est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le réveil agendé sans ambiguïté

Retourne :

un objet de classe `YWakeUpSchedule` qui permet ensuite de contrôler le réveil agendé.

YWakeUpSchedule.FirstWakeUpSchedule() yFirstWakeUpSchedule()yFirstWakeUpSchedule()

YWakeUpSchedule

Commence l'énumération des réveils agendés accessibles par la librairie.

```
function yFirstWakeUpSchedule( )
```

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule()` pour itérer sur les autres réveils agendés.

Retourne :

un pointeur sur un objet `YWakeUpSchedule`, correspondant au premier réveil agendé accessible en ligne, ou `null` si il n'y a pas de réveils agendés disponibles.

**wakeupschedule→describe()
wakeupschedule.describe()****YWakeUpSchedule**

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a 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(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le réveil agendé (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

wakeupschedule→**get_advertisedValue()**

YWakeUpSchedule

wakeupschedule→**advertisedValue()**

wakeupschedule.get_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du réveil agendé (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

wakeupschedule→**get_errorMessage()****YWakeUpSchedule****wakeupschedule**→**errorMessage()****wakeupschedule**.**get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

wakeupschedule→get_errorType()

YWakeUpSchedule

wakeupschedule→errorType()

wakeupschedule.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

wakeupschedule→get_friendlyName()**YWakeUpSchedule****wakeupschedule→friendlyName()****wakeupschedule.get_friendlyName()**

Retourne un identifiant global du réveil agendé au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du réveil agendé si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du réveil agendé (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le réveil agendé en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

wakeupschedule→**get_functionDescriptor()**

YWakeUpSchedule

wakeupschedule→**functionDescriptor()**

wakeupschedule.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

wakeupschedule→get_functionId()**YWakeUpSchedule****wakeupschedule→functionId()****wakeupschedule.get_functionId()**

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le réveil agendé (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

wakeupschedule→**get_hardwareId()**

YWakeUpSchedule

wakeupschedule→**hardwareId()**

wakeupschedule.get_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du réveil agendé (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le réveil agendé (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

wakeupschedule→get_hours()**YWakeUpSchedule****wakeupschedule→hours()****wakeupschedule.get_hours()**

Retourne les heures où le réveil est actif..

```
function get_hours( )
```

Retourne :

un entier représentant les heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_HOURS_INVALID`.

`wakeupschedule`→`get_logicalName()`

YWakeUpSchedule

`wakeupschedule`→`logicalName()`

`wakeupschedule.get_logicalName()`

Retourne le nom logique du réveil agendé.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du réveil agendé. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

wakeupschedule→get_minutes()

YWakeUpSchedule

wakeupschedule→minutes()

wakeupschedule.get_minutes()

Retourne toutes les minutes de chaque heure où le réveil est actif.

```
function get_minutes( )
```

`wakeupschedule`→`get_minutesA()`

YWakeUpSchedule

`wakeupschedule`→`minutesA()`

`wakeupschedule.get_minutesA()`

Retourne les minutes de l'interval 00-29 de chaque heures où le réveil est actif.

```
function get_minutesA( )
```

Retourne :

un entier représentant les minutes de l'interval 00-29 de chaque heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_MINUTESA_INVALID`.

wakeupschedule→get_minutesB()**YWakeUpSchedule****wakeupschedule→minutesB()****wakeupschedule.get_minutesB()**

Retourne les minutes de l'intervall 30-59 de chaque heure où le réveil est actif.

```
function get_minutesB( )
```

Retourne :

un entier représentant les minutes de l'intervall 30-59 de chaque heure où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_MINUTESB_INVALID`.

wakeupschedule→**get_module()**

YWakeUpSchedule

wakeupschedule→**module()**

wakeupschedule.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

wakeupschedule→**get_module_async()****YWakeUpSchedule****wakeupschedule**→**module_async()****wakeupschedule.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupschedule→get_monthDays()

YWakeUpSchedule

wakeupschedule→monthDays()

wakeupschedule.get_monthDays()

Retourne les jours du mois où le réveil est actif..

```
function get_monthDays( )
```

Retourne :

un entier représentant les jours du mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_MONTHDAYS_INVALID.

wakeupschedule→get_months()**YWakeUpSchedule****wakeupschedule→months()****wakeupschedule.get_months()**

Retourne les mois où le réveil est actif..

```
function get_months( )
```

Retourne :

un entier représentant les mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_MONTHS_INVALID`.

`wakeupschedule`→`get_nextOccurence()`

`YWakeUpSchedule`

`wakeupschedule`→`nextOccurence()`

`wakeupschedule.get_nextOccurence()`

Retourne la date/heure de la prochaine occurrence de réveil

```
function get_nextOccurence() ( )
```

Retourne :

un entier représentant la date/heure de la prochaine occurrence 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( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

`wakeupschedule`→`get_weekDays()`

`YWakeUpSchedule`

`wakeupschedule`→`weekDays()`

`wakeupschedule.get_weekDays()`

Retourne les jours de la semaine où le réveil est actif..

```
function get_weekDays()
```

Retourne :

un entier représentant les jours de la semaine où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_WEEKDAYS_INVALID`.

**wakeupschedule→isOnline()
wakeupschedule.isOnline()**

YWakeUpSchedule

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du réveil agendé sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le réveil agendé est joignable, `false` sinon

wakeupschedule→**isOnline_async()**
wakeupschedule.isOnline_async()

YWakeUpSchedule

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du réveil agendé sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupschedule→load()wakeupschedule.load()**YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→load_async() wakeupschedule.load_async()

YWakeUpSchedule

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wakeupschedule→**nextWakeUpSchedule()**
wakeupschedule.nextWakeUpSchedule()

YWakeUpSchedule

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

```
function nextWakeUpSchedule() ( )
```

Retourne :

un pointeur sur un objet `YWakeUpSchedule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

wakeupschedule→**registerValueCallback()**
wakeupschedule.registerValueCallback()

YWakeUpSchedule

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wakeupschedule→set_hours()**YWakeUpSchedule****wakeupschedule→setHours()****wakeupschedule.set_hours()**

Modifie les heures où un réveil doit avoir lieu

```
function set_hours( newval)
```

Paramètres :

newval un entier représentant les heures où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`wakeupschedule`→`set_logicalName()`

YWakeUpSchedule

`wakeupschedule`→`setLogicalName()`

`wakeupschedule.set_logicalName()`

Modifie le nom logique du réveil agendé.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du réveil agendé.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_minutes()****YWakeUpSchedule****wakeupschedule**→**setMinutes()****wakeupschedule.set_minutes()**

Modifie toutes les minutes où un réveil doit avoir lieu

```
function set_minutes( bitmap)
```

Paramètres :

bitmap Minutes 00-59 de chaque heure où le réveil est actif.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`wakeupschedule`→`set_minutesA()`

`YWakeUpSchedule`

`wakeupschedule`→`setMinutesA()`

`wakeupschedule.set_minutesA()`

Modifie les minutes de l'interval 00-29 où un réveil doit avoir lieu

```
function set_minutesA( newval)
```

Paramètres :

newval un entier représentant les minutes de l'interval 00-29 où un réveil doit avoir lieu

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_minutesB()****YWakeUpSchedule****wakeupschedule**→**setMinutesB()****wakeupschedule.set_minutesB()**

Modifie les minutes de l'intervall 30-59 où un réveil doit avoir lieu.

```
function set_minutesB( newval)
```

Paramètres :

newval un entier représentant les minutes de l'intervall 30-59 où un réveil doit avoir lieu

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_monthDays()**

YWakeUpSchedule

wakeupschedule→**setMonthDays()**

wakeupschedule.set_monthDays()

Modifie les jours du mois où un réveil doit avoir lieu

```
function set_monthDays( newval)
```

Paramètres :

newval un entier représentant les jours du mois où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_months()****YWakeUpSchedule****wakeupschedule**→**setMonths()****wakeupschedule.set_months()**

Modifie les mois où un réveil doit avoir lieu

```
function set_months( newval)
```

Paramètres :

newval un entier représentant les mois où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_userdata()**

YWakeUpSchedule

wakeupschedule→**setUserData()**

wakeupschedule.set_userdata()

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

wakeupschedule→**set_weekDays()****YWakeUpSchedule****wakeupschedule**→**setWeekDays()****wakeupschedule.set_weekDays()**

Modifie les jours de la semaine où un réveil doit avoir lieu

```
function set_weekDays( newval)
```

Paramètres :

newval un entier représentant les jours de la semaine où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**wait_async()**
wakeupschedule.wait_async()

YWakeUpSchedule

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.45. Interface de la fonction Watchdog

La fonction WatchDog est gérée comme un relais qui couperait brièvement l'alimentation d'un appareil après un d'attente temps donné afin de provoquer une réinitialisation complète de cet appareil. Il suffit d'appeler le watchdog à intervalle régulier pour l'empêcher de provoquer la réinitialisation. Le watchdog peut aussi être piloté directement à l'aide des méthode *pulse* et *delayedpulse* pour éteindre un appareil pendant un temps donné.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_watchdog.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YWatchdog = yoctolib.YWatchdog;</code>
php	<code>require_once('yocto_watchdog.php');</code>
cpp	<code>#include "yocto_watchdog.h"</code>
m	<code>#import "yocto_watchdog.h"</code>
pas	<code>uses yocto_watchdog;</code>
vb	<code>yocto_watchdog.vb</code>
cs	<code>yocto_watchdog.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YWatchdog;</code>
py	<code>from yocto_watchdog import *</code>

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_FUNCTION`.

watchdog→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCRIPTOR` correspondant à la fonction.

watchdog→get_functionId()

Retourne l'identifiant matériel du watchdog, sans référence au module.

watchdog→**get_hardwareId()**

Retourne l'identifiant matériel unique du watchdog au format SERIAL . FUNCTIONID.

watchdog→**get_logicalName()**

Retourne le nom logique du watchdog.

watchdog→**get_maxTimeOnStateA()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

watchdog→**get_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

watchdog→**get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

watchdog→**get_module_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

watchdog→**get_output()**

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

watchdog→**get_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

watchdog→**get_running()**

Retourne l'état du watchdog.

watchdog→**get_state()**

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

watchdog→**get_stateAtPowerOn()**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

watchdog→**get_triggerDelay()**

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

watchdog→**get_triggerDuration()**

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

watchdog→**get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

watchdog→**isOnline()**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

watchdog→**isOnline_async(callback, context)**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

watchdog→**load(msValidity)**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

watchdog→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

watchdog→**nextWatchdog()**

Continue l'énumération des watchdog commencée à l'aide de yFirstWatchdog ().

watchdog→**pulse(ms_duration)**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

watchdog→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

watchdog→**resetWatchdog()**

Réinitialise le WatchDog.

watchdog→**set_autoStart(newval)**

Modifie l'état du watching au démarrage du module.

watchdog→**set_logicalName(newval)**

Modifie le nom logique du watchdog.

watchdog→**set_maxTimeOnStateA(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

watchdog→**set_maxTimeOnStateB(newval)**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

watchdog→**set_output(newval)**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

watchdog→**set_running(newval)**

Modifie manuellement l'état de fonctionnement du watchdog.

watchdog→**set_state(newval)**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

watchdog→**set_stateAtPowerOn(newval)**

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

watchdog→**set_triggerDelay(newval)**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

watchdog→**set_triggerDuration(newval)**

Modifie la durée des resets générés par le watchdog, en millisecondes.

watchdog→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

watchdog→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWatchdog.FindWatchdog() yFindWatchdog()yFindWatchdog()

YWatchdog

Permet de retrouver un watchdog d'après un identifiant donné.

```
function yFindWatchdog( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le watchdog soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWatchdog.isOnline()` pour tester si le watchdog est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le watchdog sans ambiguïté

Retourne :

un objet de classe `YWatchdog` qui permet ensuite de contrôler le watchdog.

**YWatchdog.FirstWatchdog()
yFirstWatchdog()yFirstWatchdog()**

YWatchdog

Commence l'énumération des watchdog accessibles par la librairie.

```
function yFirstWatchdog( )
```

Utiliser la fonction `YWatchdog.nextWatchdog()` pour itérer sur les autres watchdog.

Retourne :

un pointeur sur un objet `YWatchdog`, correspondant au premier watchdog accessible en ligne, ou `null` si il n'y a pas de watchdog disponibles.

watchdog → **delayedPulse()** **watchdog.delayedPulse()**

YWatchdog

Pré-programme une impulsion

```
function delayedPulse( ms_delay, ms_duration)
```

Paramètres :

ms_delay délai d'attente avant l'impulsion, en millisecondes

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→describe()watchdog.describe()**YWatchdog**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le watchdog (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

watchdog→**get_advertisedValue()**

YWatchdog

watchdog→**advertisedValue()**

watchdog.get_advertisedValue()

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante du watchdog (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

watchdog→**get_autoStart()****YWatchdog****watchdog**→**autoStart()****watchdog.get_autoStart()**

Retourne l'état du watchdog à la mise sous tension du module.

```
function get_autoStart( )
```

Retourne :

soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watchdog à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_AUTOSTART_INVALID`.

watchdog→**get_countdown()**

YWatchdog

watchdog→**countdown()****watchdog.get_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

```
function get_countdown( )
```

Si aucune impulsion n'est programmée, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

watchdog→get_errorMessage()**YWatchdog****watchdog→errorMessage()****watchdog.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

watchdog→**get_errorType()**

YWatchdog

watchdog→**errorType()****watchdog.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

watchdog→**get_friendlyName()****YWatchdog****watchdog**→**friendlyName()****watchdog.get_friendlyName()**

Retourne un identifiant global du watchdog au format `NOM_MODULE . NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du watchdog si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du watchdog (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le watchdog en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

watchdog→**get_functionDescriptor()**

YWatchdog

watchdog→**functionDescriptor()**

watchdog.get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

watchdog→**get_functionId()****YWatchdog****watchdog**→**functionId()****watchdog.get_functionId()**

Retourne l'identifiant matériel du watchdog, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le watchdog (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

watchdog→**get_hardwareId()**

YWatchdog

watchdog→**hardwareId()****watchdog.get_hardwareId()**

Retourne l'identifiant matériel unique du watchdog au format `SERIAL.FUNCTIONID`.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du watchdog (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le watchdog (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

watchdog→get_logicalName()**YWatchdog****watchdog→logicalName()****watchdog.get_logicalName()**

Retourne le nom logique du watchdog.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique du watchdog. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

watchdog→**get_maxTimeOnStateA()**

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( )
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEA_INVALID`.

watchdog→**get_maxTimeOnStateB()****YWatchdog****watchdog**→**maxTimeOnStateB()****watchdog.get_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
function get_maxTimeOnStateB( )
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEB_INVALID`.

watchdog→**get_module()**

YWatchdog

watchdog→**module()****watchdog.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

watchdog→**get_module_async()****YWatchdog****watchdog**→**module_async()****watchdog.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

watchdog→**get_output()**

YWatchdog

watchdog→**output()****watchdog.get_output()**

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
function get_output( )
```

Retourne :

soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUT_INVALID`.

watchdog→get_pulseTimer()**YWatchdog****watchdog→pulseTimer()watchdog.get_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
function get_pulseTimer( )
```

Si aucune impulsion n'est en cours, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne `Y_PULSETIMER_INVALID`.

watchdog→**get_running()**

YWatchdog

watchdog→**running()****watchdog.get_running()**

Retourne l'état du watchdog.

```
function get_running( )
```

Retourne :

soit `Y_RUNNING_OFF`, soit `Y_RUNNING_ON`, selon l'état du watchdog

En cas d'erreur, déclenche une exception ou retourne `Y_RUNNING_INVALID`.

watchdog→**get_state()****YWatchdog****watchdog**→**state()****watchdog.get_state()**

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
function get_state( )
```

Retourne :

soit `Y_STATE_A`, soit `Y_STATE_B`, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne `Y_STATE_INVALID`.

watchdog→**get_stateAtPowerOn()**

YWatchdog

watchdog→**stateAtPowerOn()**

watchdog.get_stateAtPowerOn()

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function get_stateAtPowerOn( )
```

Retourne :

une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B` représentant l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne `Y_STATEATPOWERON_INVALID`.

watchdog→get_triggerDelay()**YWatchdog****watchdog→triggerDelay()****watchdog.get_triggerDelay()**

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

```
function get_triggerDelay( )
```

Retourne :

un entier représentant le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne `Y_TRIGGERDELAY_INVALID`.

watchdog→**get_triggerDuration()**

YWatchdog

watchdog→**triggerDuration()**

watchdog.get_triggerDuration()

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

```
function get_triggerDuration( )
```

Retourne :

un entier représentant la durée d'un reset généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne `Y_TRIGGERDURATION_INVALID`.

watchdog→**get_userdata()****YWatchdog****watchdog**→**userData()****watchdog.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

watchdog→isOnline()watchdog.isOnline()

YWatchdog

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du watchdog sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le watchdog est joignable, `false` sinon

watchdog→isOnline_async()
watchdog.isOnline_async()

YWatchdog

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du watchdog sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

watchdog→load()**watchdog.load()**

YWatchdog

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→load_async()**watchdog.load_async()****YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

watchdog→**nextWatchdog()**
watchdog.nextWatchdog()

YWatchdog

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

```
function nextWatchdog( )
```

Retourne :

un pointeur sur un objet `YWatchdog` accessible en ligne, ou `null` lorsque l'énumération est terminée.

watchdog→pulse()watchdog.pulse()**YWatchdog**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
function pulse( ms_duration)
```

Paramètres :

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**registerValueCallback()**
watchdog.registerValueCallback()

YWatchdog

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

**watchdog→resetWatchdog()
watchdog.resetWatchdog()**

YWatchdog

Réinitialise le WatchDog.

```
function resetWatchdog( )
```

Quand le watchdog est en fonctionnement cette fonction doit être appelée à interval régulier, pour empêcher que le watdog ne se déclenche

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_autoStart()**

YWatchdog

watchdog→**setAutoStart()****watchdog.set_autoStart()**

Modifie l'état du watching au démarrage du module.

```
function set_autoStart( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watching au démarrage du module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_logicalName()**YWatchdog****watchdog→setLogicalName()****watchdog.set_logicalName()**

Modifie le nom logique du watchdog.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du watchdog.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_maxTimeOnStateA()**

YWatchdog

watchdog→**setMaxTimeOnStateA()**

watchdog.set_maxTimeOnStateA()

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
function set_maxTimeOnStateA( newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_maxTimeOnStateB()****YWatchdog****watchdog**→**setMaxTimeOnStateB()****watchdog.set_maxTimeOnStateB()**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
function set_maxTimeOnStateB( newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_output()**

YWatchdog

watchdog→**setOutput()****watchdog.set_output()**

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
function set_output( newval)
```

Paramètres :

newval soit Y_OUTPUT_OFF, soit Y_OUTPUT_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_running()****YWatchdog****watchdog**→**setRunning()****watchdog.set_running()**

Modifie manuellement l'état de fonctionnement du watchdog.

```
function set_running( newval)
```

Paramètres :

newval soit `Y_RUNNING_OFF`, soit `Y_RUNNING_ON`, selon manuellement l'état de fonctionnement du watchdog

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_state()**

YWatchdog

watchdog→**setState()****watchdog.set_state()**

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
function set_state( newval)
```

Paramètres :

newval soit Y_STATE_A, soit Y_STATE_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_stateAtPowerOn()**YWatchdog****watchdog→setStateAtPowerOn()****watchdog.set_stateAtPowerOn()**

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
function set_stateAtPowerOn( newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_triggerDelay()**

YWatchdog

watchdog→**setTriggerDelay()**

watchdog.set_triggerDelay()

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

```
function set_triggerDelay( newval)
```

Paramètres :

newval un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_triggerDuration()****YWatchdog****watchdog**→**setTriggerDuration()****watchdog.set_triggerDuration()**

Modifie la durée des resets générés par le watchdog, en millisecondes.

```
function set_triggerDuration( newval)
```

Paramètres :

newval un entier représentant la durée des resets générés par le watchdog, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_userdata()**

YWatchdog

watchdog→**setUserData()****watchdog.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

watchdog→**wait_async()****watchdog.wait_async()****YWatchdog**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

3.46. Interface de la fonction Wireless

La fonction YWireless permet de configurer et de contrôler la configuration du réseau sans fil sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wireless.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWireless = yoctolib.YWireless;
php	require_once('yocto_wireless.php');
c++	#include "yocto_wireless.h"
m	#import "yocto_wireless.h"
pas	uses yocto_wireless;
vb	yocto_wireless.vb
cs	yocto_wireless.cs
java	import com.yoctopuce.YoctoAPI.YWireless;
py	from yocto_wireless import *

Fonction globales

yFindWireless(func)

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

yFirstWireless()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

Méthodes des objets YWireless

wireless→adhocNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

wireless→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE (NAME) = SERIAL . FUNCTIONID.

wireless→get_advertisedValue()

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

wireless→get_channel()

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

wireless→get_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

wireless→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

wireless→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

wireless→get_friendlyName()

Retourne un identifiant global de l'interface réseau sans fil au format NOM_MODULE . NOM_FONCTION.

wireless→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

wireless→get_functionId()

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

wireless→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL . FUNCTIONID.

wireless→get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

wireless→get_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

wireless→get_message()

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

wireless→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

wireless→get_ssid()

Retourne le nom (SSID) du réseau sans-fil sélectionné.

wireless→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

wireless→isOnline()

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

wireless→isOnline_async(callback, context)

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

wireless→joinNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

wireless→load(msValidity)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

wireless→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

wireless→nextWireless()

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de yFirstWireless().

wireless→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wireless→set_logicalName(newval)

Modifie le nom logique de l'interface réseau sans fil.

wireless→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userData.

wireless→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWireless.FindWireless() yFindWireless()yFindWireless()

YWireless

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

```
function yFindWireless( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau sans fil soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWireless.isOnline()` pour tester si l'interface réseau sans fil est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'interface réseau sans fil sans ambiguïté

Retourne :

un objet de classe `YWireless` qui permet ensuite de contrôler l'interface réseau sans fil.

**YWireless.FirstWireless()
yFirstWireless(yFirstWireless())**

YWireless

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

```
function yFirstWireless( )
```

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

Retourne :

un pointeur sur un objet `YWireless`, correspondant à la première interface réseau sans fil accessible en ligne, ou `null` si il n'y a pas de interfaces réseau sans fil disponibles.

wireless→**adhocNetwork()****wireless.adhocNetwork()**

YWireless

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

```
function adhocNetwork( ssid, securityKey)
```

Si une clef d'accès est spécifiée, le réseau sera protégé par une sécurité WEP128 (l'utilisation de WPA n'est pas standardisée en mode ad-hoc). N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid nom du réseau sans fil à créer
securityKey clé d'accès de réseau, sous forme de chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→**describe()****wireless.describe()****YWireless**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

```
function describe( )
```

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'interface réseau sans fil (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

wireless→**get_advertisedValue()**

YWireless

wireless→**advertisedValue()**

wireless.get_advertisedValue()

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

```
function get_advertisedValue( )
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères). En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

wireless→**get_channel()****YWireless****wireless**→**channel()****wireless.get_channel()**

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

```
function get_channel( )
```

Retourne :

un entier représentant le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé

En cas d'erreur, déclenche une exception ou retourne `Y_CHANNEL_INVALID`.

wireless→**get_detectedWlans()**

YWireless

wireless→**detectedWlans()**

wireless.get_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

```
function get_detectedWlans( )
```

La liste n'est pas mise à jour quand le module est déjà connecté à un accès sans fil (mode "infrastructure"). Pour forcer la détection des réseaux sans fil, il faut appeler `adhocNetwork()` 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()****YWireless****wireless**→**errorMessage()****wireless.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

```
function get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

wireless→**get_errorType()**

YWireless

wireless→**errorType()****wireless.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

wireless→**get_friendlyName()****YWireless****wireless**→**friendlyName()****wireless.get_friendlyName()**

Retourne un identifiant global de l'interface réseau sans fil au format `NOM_MODULE.NOM_FONCTION`.

```
function get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de l'interface réseau sans fil si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau sans fil (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'interface réseau sans fil en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

wireless→**get_functionDescriptor()**
wireless→**functionDescriptor()**
wireless.get_functionDescriptor()

YWireless

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

```
function get_functionDescriptor( )
```

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`. Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

wireless→**get_functionId()****YWireless****wireless**→**functionId()****wireless.get_functionId()**

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

```
function get_functionId( )
```

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'interface réseau sans fil (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

wireless→**get_hardwareId()**

YWireless

wireless→**hardwareId()****wireless.get_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format SERIAL.FUNCTIONID.

```
function get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau sans fil (par exemple RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'interface réseau sans fil (ex: RELAYLO1-123456.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 connexion, exprimée en pourcents.

```
function get_linkQuality( )
```

Retourne :

un entier représentant la qualité de la connexion, exprimée en pourcents

En cas d'erreur, déclenche une exception ou retourne `Y_LINKQUALITY_INVALID`.

wireless→**get_logicalName()**

YWireless

wireless→**logicalName()****wireless.get_logicalName()**

Retourne le nom logique de l'interface réseau sans fil.

```
function get_logicalName( )
```

Retourne :

une chaîne de caractères représentant le nom logique de l'interface réseau sans fil. En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

wireless→**get_message()****YWireless****wireless**→**message()****wireless.get_message()**

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

```
function get_message( )
```

Retourne :

une chaîne de caractères représentant le dernier message de diagnostic de l'interface au réseau sans fil

En cas d'erreur, déclenche une exception ou retourne `Y_MESSAGE_INVALID`.

wireless→**get_module()**

YWireless

wireless→**module()****wireless.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

wireless→**get_module_async()****YWireless****wireless**→**module_async()****wireless.get_module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wireless→**get_security()**

YWireless

wireless→**security()****wireless.get_security()**

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

```
function get_security( )
```

Retourne :

une valeur parmi Y_SECURITY_UNKNOWN, Y_SECURITY_OPEN, Y_SECURITY_WEP, Y_SECURITY_WPA et Y_SECURITY_WPA2 représentant l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y_SECURITY_INVALID.

wireless→**get_ssid()****YWireless****wireless**→**ssid()****wireless.get_ssid()**

Retourne le nom (SSID) du réseau sans-fil sélectionné.

```
function get_ssid( )
```

Retourne :

une chaîne de caractères représentant le nom (SSID) du réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne `Y_SSID_INVALID`.

wireless→**get_userdata()**

YWireless

wireless→**userData()****wireless.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

```
function get_userdata( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

wireless→**isOnline()****wireless.isOnline()****YWireless**

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'interface réseau sans fil sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'interface réseau sans fil est joignable, `false` sinon

wireless→**isOnline_async()****wireless.isOnline_async()**

YWireless

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

```
function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'interface réseau sans fil sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wireless→**joinNetwork()****wireless.joinNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

```
function joinNetwork( ssid, securityKey)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid nom du réseau sans fil à utiliser

securityKey clé d'accès au réseau, sous forme de chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→**load()****wireless.load()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
function load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→**load_async()****wireless.load_async()****YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

wireless→**nextWireless()****wireless.nextWireless()**

YWireless

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

```
function nextWireless()
```

Retourne :

un pointeur sur un objet `YWireless` accessible en ligne, ou `null` lorsque l'énumération est terminée.

wireless→**registerValueCallback()**
wireless.registerValueCallback()

YWireless

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
function registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wireless→**set_logicalName()**

YWireless

wireless→**setLogicalName()**

wireless.set_logicalName()

Modifie le nom logique de l'interface réseau sans fil.

```
function set_logicalName( newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→**set_userdata()****YWireless****wireless**→**setUserData()****wireless.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
function set_userdata( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

wireless→wait_async()wireless.wait_async()

YWireless

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout :

Index

A

Accelerometer 32
adhocNetwork, YWireless 1723
Alimentation 487
AnButton 78

B

Blueprint 12
Brute 341

C

calibrate, YLightSensor 761
calibrateFromPoints, YAccelerometer 36
calibrateFromPoints, YCarbonDioxide 124
calibrateFromPoints, YCompass 200
calibrateFromPoints, YCurrent 244
calibrateFromPoints, YGenericSensor 553
calibrateFromPoints, YGyro 603
calibrateFromPoints, YHumidity 687
calibrateFromPoints, YLightSensor 762
calibrateFromPoints, YMagnetometer 805
calibrateFromPoints, YPower 990
calibrateFromPoints, YPressure 1037
calibrateFromPoints, YQt 1149
calibrateFromPoints, YSensor 1303
calibrateFromPoints, YTemperature 1385
calibrateFromPoints, YTilt 1430
calibrateFromPoints, YVoc 1473
calibrateFromPoints, YVoltage 1516
callbackLogin, YNetwork 903
cancel3DCalibration, YRefFrame 1223
CarbonDioxide 120
CheckLogicalName, YAPI 14
clear, YDisplayLayer 456
clearConsole, YDisplayLayer 457
ColorLed 163
Compass 196
Configuration 1219
consoleOut, YDisplayLayer 458
Contrôle 3, 5, 487, 853, 959
copyLayerContent, YDisplay 408
Current 240

D

DataLogger 283
delayedPulse, YDigitalIO 360
delayedPulse, YRelay 1263
delayedPulse, YWatchdog 1675
describe, YAccelerometer 37
describe, YAnButton 82
describe, YCarbonDioxide 125
describe, YColorLed 166

describe, YCompass 201
describe, YCurrent 245
describe, YDataLogger 286
describe, YDigitalIO 361
describe, YDisplay 409
describe, YDualPower 490
describe, YFiles 519
describe, YGenericSensor 554
describe, YGyro 604
describe, YHubPort 657
describe, YHumidity 688
describe, YLed 729
describe, YLightSensor 763
describe, YMagnetometer 806
describe, YModule 857
describe, YNetwork 904
describe, YOsControl 962
describe, YPower 991
describe, YPressure 1038
describe, YPwmOutput 1080
describe, YPwmPowerSource 1121
describe, YQt 1150
describe, YRealTimeClock 1191
describe, YRefFrame 1224
describe, YRelay 1264
describe, YSensor 1304
describe, YServo 1346
describe, YTemperature 1386
describe, YTilt 1431
describe, YVoc 1474
describe, YVoltage 1517
describe, YVSource 1558
describe, YWakeUpMonitor 1595
describe, YWakeUpSchedule 1634
describe, YWatchdog 1676
describe, YWireless 1724
DigitalIO 356
DisableExceptions, YAPI 15
Display 404
DisplayLayer 455
Données 318, 328, 341
download, YFiles 520
download, YModule 858
download_async, YFiles 521
drawBar, YDisplayLayer 459
drawBitmap, YDisplayLayer 460
drawCircle, YDisplayLayer 461
drawDisc, YDisplayLayer 462
drawImage, YDisplayLayer 463
drawPixel, YDisplayLayer 464
drawRect, YDisplayLayer 465
drawText, YDisplayLayer 466
dutyCycleMove, YPwmOutput 1081

E

EnableExceptions, YAPI 16
Enregistrées 328, 341
Erreurs 8

F

fade, YDisplay 410
Files 516
FindAccelerometer, YAccelerometer 34
FindAnButton, YAnButton 80
FindCarbonDioxide, YCarbonDioxide 122
FindColorLed, YColorLed 164
FindCompass, YCompass 198
FindCurrent, YCurrent 242
FindDataLogger, YDataLogger 284
FindDigitalIO, YDigitalIO 358
FindDisplay, YDisplay 406
FindDualPower, YDualPower 488
FindFiles, YFiles 517
FindGenericSensor, YGenericSensor 551
FindGyro, YGyro 601
FindHubPort, YHubPort 655
FindHumidity, YHumidity 685
FindLed, YLed 727
FindLightSensor, YLightSensor 759
FindMagnetometer, YMagnetometer 803
FindModule, YModule 855
FindNetwork, YNetwork 901
FindOsControl, YOsControl 960
FindPower, YPower 988
FindPressure, YPressure 1035
FindPwmOutput, YPwmOutput 1078
FindPwmPowerSource, YPwmPowerSource 1119
FindQt, YQt 1147
FindRealTimeClock, YRealTimeClock 1189
FindRefFrame, YRefFrame 1221
FindRelay, YRelay 1261
FindSensor, YSensor 1301
FindServo, YServo 1344
FindTemperature, YTemperature 1383
FindTilt, YTilt 1428
FindVoc, YVoc 1471
FindVoltage, YVoltage 1514
FindVSource, YVSource 1556
FindWakeUpMonitor, YWakeUpMonitor 1593
FindWakeUpSchedule, YWakeUpSchedule 1632
FindWatchdog, YWatchdog 1673
FindWireless, YWireless 1721
FirstAccelerometer, YAccelerometer 35
FirstAnButton, YAnButton 81
FirstCarbonDioxide, YCarbonDioxide 123
FirstColorLed, YColorLed 165
FirstCompass, YCompass 199
FirstCurrent, YCurrent 243
FirstDataLogger, YDataLogger 285
FirstDigitalIO, YDigitalIO 359
FirstDisplay, YDisplay 407

FirstDualPower, YDualPower 489
FirstFiles, YFiles 518
FirstGenericSensor, YGenericSensor 552
FirstGyro, YGyro 602
FirstHubPort, YHubPort 656
FirstHumidity, YHumidity 686
FirstLed, YLed 728
FirstLightSensor, YLightSensor 760
FirstMagnetometer, YMagnetometer 804
FirstModule, YModule 856
FirstNetwork, YNetwork 902
FirstOsControl, YOsControl 961
FirstPower, YPower 989
FirstPressure, YPressure 1036
FirstPwmOutput, YPwmOutput 1079
FirstPwmPowerSource, YPwmPowerSource 1120
FirstQt, YQt 1148
FirstRealTimeClock, YRealTimeClock 1190
FirstRefFrame, YRefFrame 1222
FirstRelay, YRelay 1262
FirstSensor, YSensor 1302
FirstServo, YServo 1345
FirstTemperature, YTemperature 1384
FirstTilt, YTilt 1429
FirstVoc, YVoc 1472
FirstVoltage, YVoltage 1515
FirstVSource, YVSource 1557
FirstWakeUpMonitor, YWakeUpMonitor 1594
FirstWakeUpSchedule, YWakeUpSchedule 1633
FirstWatchdog, YWatchdog 1674
FirstWireless, YWireless 1722
Fonctions 13, 1299
forgetAllDataStreams, YDataLogger 287
format_fs, YFiles 522
Forme 318
FreeAPI, YAPI 17
functionCount, YModule 859
functionId, YModule 860
functionName, YModule 861
functionValue, YModule 862

G

GenericSensor 549
get_3DCalibrationHint, YRefFrame 1225
get_3DCalibrationLogMsg, YRefFrame 1226
get_3DCalibrationProgress, YRefFrame 1227
get_3DCalibrationStage, YRefFrame 1228
get_3DCalibrationStageProgress, YRefFrame 1229
get_adminPassword, YNetwork 905
get_advertisedValue, YAccelerometer 38
get_advertisedValue, YAnButton 83
get_advertisedValue, YCarbonDioxide 126
get_advertisedValue, YColorLed 167
get_advertisedValue, YCompass 202
get_advertisedValue, YCurrent 246
get_advertisedValue, YDataLogger 288
get_advertisedValue, YDigitalIO 362

get_advertisedValue, YDisplay 411
get_advertisedValue, YDualPower 491
get_advertisedValue, YFiles 523
get_advertisedValue, YGenericSensor 555
get_advertisedValue, YGyro 605
get_advertisedValue, YHubPort 658
get_advertisedValue, YHumidity 689
get_advertisedValue, YLed 730
get_advertisedValue, YLightSensor 764
get_advertisedValue, YMagnetometer 807
get_advertisedValue, YNetwork 906
get_advertisedValue, YOsControl 963
get_advertisedValue, YPower 992
get_advertisedValue, YPressure 1039
get_advertisedValue, YPwmOutput 1082
get_advertisedValue, YPwmPowerSource 1122
get_advertisedValue, YQt 1151
get_advertisedValue, YRealTimeClock 1192
get_advertisedValue, YRefFrame 1230
get_advertisedValue, YRelay 1265
get_advertisedValue, YSensor 1305
get_advertisedValue, YServo 1347
get_advertisedValue, YTemperature 1387
get_advertisedValue, YTilt 1432
get_advertisedValue, YVoc 1475
get_advertisedValue, YVoltage 1518
get_advertisedValue, YVSource 1559
get_advertisedValue, YWakeUpMonitor 1596
get_advertisedValue, YWakeUpSchedule 1635
get_advertisedValue, YWatchdog 1677
get_advertisedValue, YWireless 1725
get_analogCalibration, YAnButton 84
get_autoStart, YDataLogger 289
get_autoStart, YWatchdog 1678
get_averageValue, YDataRun 318
get_averageValue, YDataStream 342
get_averageValue, YMeasure 847
get_baudRate, YHubPort 659
get_beacon, YModule 863
get_bearing, YRefFrame 1231
get_bitDirection, YDigitalIO 363
get_bitOpenDrain, YDigitalIO 364
get_bitPolarity, YDigitalIO 365
get_bitState, YDigitalIO 366
get_blinking, YLed 731
get_brightness, YDisplay 412
get_calibratedValue, YAnButton 85
get_calibrationMax, YAnButton 86
get_calibrationMin, YAnButton 87
get_callbackCredentials, YNetwork 907
get_callbackEncoding, YNetwork 908
get_callbackMaxDelay, YNetwork 909
get_callbackMethod, YNetwork 910
get_callbackMinDelay, YNetwork 911
get_callbackUrl, YNetwork 912
get_channel, YWireless 1726
get_columnCount, YDataStream 343
get_columnNames, YDataStream 344
get_cosPhi, YPower 993
get_countdown, YRelay 1266
get_countdown, YWatchdog 1679
get_currentRawValue, YAccelerometer 39
get_currentRawValue, YCarbonDioxide 127
get_currentRawValue, YCompass 203
get_currentRawValue, YCurrent 247
get_currentRawValue, YGenericSensor 556
get_currentRawValue, YGyro 606
get_currentRawValue, YHumidity 690
get_currentRawValue, YLightSensor 765
get_currentRawValue, YMagnetometer 808
get_currentRawValue, YPower 994
get_currentRawValue, YPressure 1040
get_currentRawValue, YQt 1152
get_currentRawValue, YSensor 1306
get_currentRawValue, YTemperature 1388
get_currentRawValue, YTilt 1433
get_currentRawValue, YVoc 1476
get_currentRawValue, YVoltage 1519
get_currentRunIndex, YDataLogger 290
get_currentValue, YAccelerometer 40
get_currentValue, YCarbonDioxide 128
get_currentValue, YCompass 204
get_currentValue, YCurrent 248
get_currentValue, YGenericSensor 557
get_currentValue, YGyro 607
get_currentValue, YHumidity 691
get_currentValue, YLightSensor 766
get_currentValue, YMagnetometer 809
get_currentValue, YPower 995
get_currentValue, YPressure 1041
get_currentValue, YQt 1153
get_currentValue, YSensor 1307
get_currentValue, YTemperature 1389
get_currentValue, YTilt 1434
get_currentValue, YVoc 1477
get_currentValue, YVoltage 1520
get_data, YDataStream 345
get_dataRows, YDataStream 346
get_dataSamplesIntervalMs, YDataStream 347
get_dataSets, YDataLogger 291
get_dataStreams, YDataLogger 292
get_dateTime, YRealTimeClock 1193
get_detectedWlans, YWireless 1727
get_discoverable, YNetwork 913
get_display, YDisplayLayer 467
get_displayHeight, YDisplay 413
get_displayHeight, YDisplayLayer 468
get_displayLayer, YDisplay 414
get_displayType, YDisplay 415
get_displayWidth, YDisplay 416
get_displayWidth, YDisplayLayer 469
get_duration, YDataRun 319
get_duration, YDataStream 348
get_dutyCycle, YPwmOutput 1083
get_dutyCycleAtPowerOn, YPwmOutput 1084
get_enabled, YDisplay 417
get_enabled, YHubPort 660
get_enabled, YPwmOutput 1085

get_enabled, YServo 1348
get_enabledAtPowerOn, YPwmOutput 1086
get_enabledAtPowerOn, YServo 1349
get_endTimeUTC, YDataSet 329
get_endTimeUTC, YMeasure 848
get_errorMessage, YAccelerometer 41
get_errorMessage, YAnButton 88
get_errorMessage, YCarbonDioxide 129
get_errorMessage, YColorLed 168
get_errorMessage, YCompass 205
get_errorMessage, YCurrent 249
get_errorMessage, YDataLogger 293
get_errorMessage, YDigitalIO 367
get_errorMessage, YDisplay 418
get_errorMessage, YDualPower 492
get_errorMessage, YFiles 524
get_errorMessage, YGenericSensor 558
get_errorMessage, YGyro 608
get_errorMessage, YHubPort 661
get_errorMessage, YHumidity 692
get_errorMessage, YLed 732
get_errorMessage, YLightSensor 767
get_errorMessage, YMagnetometer 810
get_errorMessage, YModule 864
get_errorMessage, YNetwork 914
get_errorMessage, YOsControl 964
get_errorMessage, YPower 996
get_errorMessage, YPressure 1042
get_errorMessage, YPwmOutput 1087
get_errorMessage, YPwmPowerSource 1123
get_errorMessage, YQt 1154
get_errorMessage, YRealTimeClock 1194
get_errorMessage, YRefFrame 1232
get_errorMessage, YRelay 1267
get_errorMessage, YSensor 1308
get_errorMessage, YServo 1350
get_errorMessage, YTemperature 1390
get_errorMessage, YTilt 1435
get_errorMessage, YVoc 1478
get_errorMessage, YVoltage 1521
get_errorMessage, YVSource 1560
get_errorMessage, YWakeUpMonitor 1597
get_errorMessage, YWakeUpSchedule 1636
get_errorMessage, YWatchdog 1680
get_errorMessage, YWireless 1728
get_errorType, YAccelerometer 42
get_errorType, YAnButton 89
get_errorType, YCarbonDioxide 130
get_errorType, YColorLed 169
get_errorType, YCompass 206
get_errorType, YCurrent 250
get_errorType, YDataLogger 294
get_errorType, YDigitalIO 368
get_errorType, YDisplay 419
get_errorType, YDualPower 493
get_errorType, YFiles 525
get_errorType, YGenericSensor 559
get_errorType, YGyro 609
get_errorType, YHubPort 662
get_errorType, YHumidity 693
get_errorType, YLed 733
get_errorType, YLightSensor 768
get_errorType, YMagnetometer 811
get_errorType, YModule 865
get_errorType, YNetwork 915
get_errorType, YOsControl 965
get_errorType, YPower 997
get_errorType, YPressure 1043
get_errorType, YPwmOutput 1088
get_errorType, YPwmPowerSource 1124
get_errorType, YQt 1155
get_errorType, YRealTimeClock 1195
get_errorType, YRefFrame 1233
get_errorType, YRelay 1268
get_errorType, YSensor 1309
get_errorType, YServo 1351
get_errorType, YTemperature 1391
get_errorType, YTilt 1436
get_errorType, YVoc 1479
get_errorType, YVoltage 1522
get_errorType, YVSource 1561
get_errorType, YWakeUpMonitor 1598
get_errorType, YWakeUpSchedule 1637
get_errorType, YWatchdog 1681
get_errorType, YWireless 1729
get_extPowerFailure, YVSource 1562
get_extVoltage, YDualPower 494
get_failure, YVSource 1563
get_filesCount, YFiles 526
get_firmwareRelease, YModule 866
get_freeSpace, YFiles 527
get_frequency, YPwmOutput 1089
get_friendlyName, YAccelerometer 43
get_friendlyName, YAnButton 90
get_friendlyName, YCarbonDioxide 131
get_friendlyName, YColorLed 170
get_friendlyName, YCompass 207
get_friendlyName, YCurrent 251
get_friendlyName, YDataLogger 295
get_friendlyName, YDigitalIO 369
get_friendlyName, YDisplay 420
get_friendlyName, YDualPower 495
get_friendlyName, YFiles 528
get_friendlyName, YGenericSensor 560
get_friendlyName, YGyro 610
get_friendlyName, YHubPort 663
get_friendlyName, YHumidity 694
get_friendlyName, YLed 734
get_friendlyName, YLightSensor 769
get_friendlyName, YMagnetometer 812
get_friendlyName, YNetwork 916
get_friendlyName, YOsControl 966
get_friendlyName, YPower 998
get_friendlyName, YPressure 1044
get_friendlyName, YPwmOutput 1090
get_friendlyName, YPwmPowerSource 1125
get_friendlyName, YQt 1156
get_friendlyName, YRealTimeClock 1196

get_friendlyName, YRefFrame 1234
get_friendlyName, YRelay 1269
get_friendlyName, YSensor 1310
get_friendlyName, YServo 1352
get_friendlyName, YTemperature 1392
get_friendlyName, YTilt 1437
get_friendlyName, YVoc 1480
get_friendlyName, YVoltage 1523
get_friendlyName, YVSource 1564
get_friendlyName, YWakeUpMonitor 1599
get_friendlyName, YWakeUpSchedule 1638
get_friendlyName, YWatchdog 1682
get_friendlyName, YWireless 1730
get_functionDescriptor, YAccelerometer 44
get_functionDescriptor, YAnButton 91
get_functionDescriptor, YCarbonDioxide 132
get_functionDescriptor, YColorLed 171
get_functionDescriptor, YCompass 208
get_functionDescriptor, YCurrent 252
get_functionDescriptor, YDataLogger 296
get_functionDescriptor, YDigitalIO 370
get_functionDescriptor, YDisplay 421
get_functionDescriptor, YDualPower 496
get_functionDescriptor, YFiles 529
get_functionDescriptor, YGenericSensor 561
get_functionDescriptor, YGyro 611
get_functionDescriptor, YHubPort 664
get_functionDescriptor, YHumidity 695
get_functionDescriptor, YLed 735
get_functionDescriptor, YLightSensor 770
get_functionDescriptor, YMagnetometer 813
get_functionDescriptor, YNetwork 917
get_functionDescriptor, YOsControl 967
get_functionDescriptor, YPower 999
get_functionDescriptor, YPressure 1045
get_functionDescriptor, YPwmOutput 1091
get_functionDescriptor, YPwmPowerSource 1126
get_functionDescriptor, YQt 1157
get_functionDescriptor, YRealTimeClock 1197
get_functionDescriptor, YRefFrame 1235
get_functionDescriptor, YRelay 1270
get_functionDescriptor, YSensor 1311
get_functionDescriptor, YServo 1353
get_functionDescriptor, YTemperature 1393
get_functionDescriptor, YTilt 1438
get_functionDescriptor, YVoc 1481
get_functionDescriptor, YVoltage 1524
get_functionDescriptor, YVSource 1565
get_functionDescriptor, YWakeUpMonitor 1600
get_functionDescriptor, YWakeUpSchedule 1639
get_functionDescriptor, YWatchdog 1683
get_functionDescriptor, YWireless 1731
get_functionId, YAccelerometer 45
get_functionId, YAnButton 92
get_functionId, YCarbonDioxide 133
get_functionId, YColorLed 172
get_functionId, YCompass 209
get_functionId, YCurrent 253
get_functionId, YDataLogger 297
get_functionId, YDataSet 330
get_functionId, YDigitalIO 371
get_functionId, YDisplay 422
get_functionId, YDualPower 497
get_functionId, YFiles 530
get_functionId, YGenericSensor 562
get_functionId, YGyro 612
get_functionId, YHubPort 665
get_functionId, YHumidity 696
get_functionId, YLed 736
get_functionId, YLightSensor 771
get_functionId, YMagnetometer 814
get_functionId, YNetwork 918
get_functionId, YOsControl 968
get_functionId, YPower 1000
get_functionId, YPressure 1046
get_functionId, YPwmOutput 1092
get_functionId, YPwmPowerSource 1127
get_functionId, YQt 1158
get_functionId, YRealTimeClock 1198
get_functionId, YRefFrame 1236
get_functionId, YRelay 1271
get_functionId, YSensor 1312
get_functionId, YServo 1354
get_functionId, YTemperature 1394
get_functionId, YTilt 1439
get_functionId, YVoc 1482
get_functionId, YVoltage 1525
get_functionId, YVSource 1566
get_functionId, YWakeUpMonitor 1601
get_functionId, YWakeUpSchedule 1640
get_functionId, YWatchdog 1684
get_functionId, YWireless 1732
get_hardwareId, YAccelerometer 46
get_hardwareId, YAnButton 93
get_hardwareId, YCarbonDioxide 134
get_hardwareId, YColorLed 173
get_hardwareId, YCompass 210
get_hardwareId, YCurrent 254
get_hardwareId, YDataLogger 298
get_hardwareId, YDataSet 331
get_hardwareId, YDigitalIO 372
get_hardwareId, YDisplay 423
get_hardwareId, YDualPower 498
get_hardwareId, YFiles 531
get_hardwareId, YGenericSensor 563
get_hardwareId, YGyro 613
get_hardwareId, YHubPort 666
get_hardwareId, YHumidity 697
get_hardwareId, YLed 737
get_hardwareId, YLightSensor 772
get_hardwareId, YMagnetometer 815
get_hardwareId, YModule 867
get_hardwareId, YNetwork 919
get_hardwareId, YOsControl 969
get_hardwareId, YPower 1001
get_hardwareId, YPressure 1047
get_hardwareId, YPwmOutput 1093
get_hardwareId, YPwmPowerSource 1128

get_hardwareId, YQt 1159
get_hardwareId, YRealTimeClock 1199
get_hardwareId, YRefFrame 1237
get_hardwareId, YRelay 1272
get_hardwareId, YSensor 1313
get_hardwareId, YServo 1355
get_hardwareId, YTemperature 1395
get_hardwareId, YTilt 1440
get_hardwareId, YVoc 1483
get_hardwareId, YVoltage 1526
get_hardwareId, YVSource 1567
get_hardwareId, YWakeUpMonitor 1602
get_hardwareId, YWakeUpSchedule 1641
get_hardwareId, YWatchdog 1685
get_hardwareId, YWireless 1733
get_heading, YGyro 614
get_highestValue, YAccelerometer 47
get_highestValue, YCarbonDioxide 135
get_highestValue, YCompass 211
get_highestValue, YCurrent 255
get_highestValue, YGenericSensor 564
get_highestValue, YGyro 615
get_highestValue, YHumidity 698
get_highestValue, YLightSensor 773
get_highestValue, YMagnetometer 816
get_highestValue, YPower 1002
get_highestValue, YPressure 1048
get_highestValue, YQt 1160
get_highestValue, YSensor 1314
get_highestValue, YTemperature 1396
get_highestValue, YTilt 1441
get_highestValue, YVoc 1484
get_highestValue, YVoltage 1527
get_hours, YWakeUpSchedule 1642
get_hslColor, YColorLed 174
get_icon2d, YModule 868
get_ipAddress, YNetwork 920
get_isPressed, YAnButton 94
get_lastLogs, YModule 869
get_lastTimePressed, YAnButton 95
get_lastTimeReleased, YAnButton 96
get_layerCount, YDisplay 424
get_layerHeight, YDisplay 425
get_layerHeight, YDisplayLayer 470
get_layerWidth, YDisplay 426
get_layerWidth, YDisplayLayer 471
get_linkQuality, YWireless 1734
get_list, YFiles 532
get_logFrequency, YAccelerometer 48
get_logFrequency, YCarbonDioxide 136
get_logFrequency, YCompass 212
get_logFrequency, YCurrent 256
get_logFrequency, YGenericSensor 565
get_logFrequency, YGyro 616
get_logFrequency, YHumidity 699
get_logFrequency, YLightSensor 774
get_logFrequency, YMagnetometer 817
get_logFrequency, YPower 1003
get_logFrequency, YPressure 1049
get_logFrequency, YQt 1161
get_logFrequency, YSensor 1315
get_logFrequency, YTemperature 1397
get_logFrequency, YTilt 1442
get_logFrequency, YVoc 1485
get_logFrequency, YVoltage 1528
get_logicalName, YAccelerometer 49
get_logicalName, YAnButton 97
get_logicalName, YCarbonDioxide 137
get_logicalName, YColorLed 175
get_logicalName, YCompass 213
get_logicalName, YCurrent 257
get_logicalName, YDataLogger 299
get_logicalName, YDigitalIO 373
get_logicalName, YDisplay 427
get_logicalName, YDualPower 499
get_logicalName, YFiles 533
get_logicalName, YGenericSensor 566
get_logicalName, YGyro 617
get_logicalName, YHubPort 667
get_logicalName, YHumidity 700
get_logicalName, YLed 738
get_logicalName, YLightSensor 775
get_logicalName, YMagnetometer 818
get_logicalName, YModule 870
get_logicalName, YNetwork 921
get_logicalName, YOsControl 970
get_logicalName, YPower 1004
get_logicalName, YPressure 1050
get_logicalName, YPwmOutput 1094
get_logicalName, YPwmPowerSource 1129
get_logicalName, YQt 1162
get_logicalName, YRealTimeClock 1200
get_logicalName, YRefFrame 1238
get_logicalName, YRelay 1273
get_logicalName, YSensor 1316
get_logicalName, YServo 1356
get_logicalName, YTemperature 1398
get_logicalName, YTilt 1443
get_logicalName, YVoc 1486
get_logicalName, YVoltage 1529
get_logicalName, YVSource 1568
get_logicalName, YWakeUpMonitor 1603
get_logicalName, YWakeUpSchedule 1643
get_logicalName, YWatchdog 1686
get_logicalName, YWireless 1735
get_lowestValue, YAccelerometer 50
get_lowestValue, YCarbonDioxide 138
get_lowestValue, YCompass 214
get_lowestValue, YCurrent 258
get_lowestValue, YGenericSensor 567
get_lowestValue, YGyro 618
get_lowestValue, YHumidity 701
get_lowestValue, YLightSensor 776
get_lowestValue, YMagnetometer 819
get_lowestValue, YPower 1005
get_lowestValue, YPressure 1051
get_lowestValue, YQt 1163
get_lowestValue, YSensor 1317

get_lowestValue, YTemperature 1399
get_lowestValue, YTilt 1444
get_lowestValue, YVoc 1487
get_lowestValue, YVoltage 1530
get_luminosity, YLed 739
get_luminosity, YModule 871
get_macAddress, YNetwork 922
get_magneticHeading, YCompass 215
get_maxTimeOnStateA, YRelay 1274
get_maxTimeOnStateA, YWatchdog 1687
get_maxTimeOnStateB, YRelay 1275
get_maxTimeOnStateB, YWatchdog 1688
get_maxValue, YDataRun 320
get_maxValue, YDataStream 349
get_maxValue, YMeasure 849
get_measureNames, YDataRun 321
get_measures, YDataSet 332
get_message, YWireless 1736
get_meter, YPower 1006
get_meterTimer, YPower 1007
get_minutes, YWakeUpSchedule 1644
get_minutesA, YWakeUpSchedule 1645
get_minutesB, YWakeUpSchedule 1646
get_minValue, YDataRun 322
get_minValue, YDataStream 350
get_minValue, YMeasure 850
get_module, YAccelerometer 51
get_module, YAnButton 98
get_module, YCarbonDioxide 139
get_module, YColorLed 176
get_module, YCompass 216
get_module, YCurrent 259
get_module, YDataLogger 300
get_module, YDigitalIO 374
get_module, YDisplay 428
get_module, YDualPower 500
get_module, YFiles 534
get_module, YGenericSensor 568
get_module, YGyro 619
get_module, YHubPort 668
get_module, YHumidity 702
get_module, YLed 740
get_module, YLightSensor 777
get_module, YMagnetometer 820
get_module, YNetwork 923
get_module, YOsControl 971
get_module, YPower 1008
get_module, YPressure 1052
get_module, YPwmOutput 1095
get_module, YPwmPowerSource 1130
get_module, YQt 1164
get_module, YRealTimeClock 1201
get_module, YRefFrame 1239
get_module, YRelay 1276
get_module, YSensor 1318
get_module, YServo 1357
get_module, YTemperature 1400
get_module, YTilt 1445
get_module, YVoc 1488

get_module, YVoltage 1531
get_module, YVSource 1569
get_module, YWakeUpMonitor 1604
get_module, YWakeUpSchedule 1647
get_module, YWatchdog 1689
get_module, YWireless 1737
get_module_async, YAccelerometer 52
get_module_async, YAnButton 99
get_module_async, YCarbonDioxide 140
get_module_async, YColorLed 177
get_module_async, YCompass 217
get_module_async, YCurrent 260
get_module_async, YDataLogger 301
get_module_async, YDigitalIO 375
get_module_async, YDisplay 429
get_module_async, YDualPower 501
get_module_async, YFiles 535
get_module_async, YGenericSensor 569
get_module_async, YGyro 620
get_module_async, YHubPort 669
get_module_async, YHumidity 703
get_module_async, YLed 741
get_module_async, YLightSensor 778
get_module_async, YMagnetometer 821
get_module_async, YNetwork 924
get_module_async, YOsControl 972
get_module_async, YPower 1009
get_module_async, YPressure 1053
get_module_async, YPwmOutput 1096
get_module_async, YPwmPowerSource 1131
get_module_async, YQt 1165
get_module_async, YRealTimeClock 1202
get_module_async, YRefFrame 1240
get_module_async, YRelay 1277
get_module_async, YSensor 1319
get_module_async, YServo 1358
get_module_async, YTemperature 1401
get_module_async, YTilt 1446
get_module_async, YVoc 1489
get_module_async, YVoltage 1532
get_module_async, YVSource 1570
get_module_async, YWakeUpMonitor 1605
get_module_async, YWakeUpSchedule 1648
get_module_async, YWatchdog 1690
get_module_async, YWireless 1738
get_monthDays, YWakeUpSchedule 1649
get_months, YWakeUpSchedule 1650
get_mountOrientation, YRefFrame 1241
get_mountPosition, YRefFrame 1242
get_neutral, YServo 1359
get_nextOccurence, YWakeUpSchedule 1651
get_nextWakeUp, YWakeUpMonitor 1606
get_orientation, YDisplay 430
get_output, YRelay 1278
get_output, YWatchdog 1691
get_outputVoltage, YDigitalIO 376
get_overCurrent, YVSource 1571
get_overHeat, YVSource 1572
get_overLoad, YVSource 1573

get_period, YPwmOutput 1097
get_persistentSettings, YModule 872
get_pitch, YGyro 621
get_poeCurrent, YNetwork 925
get_portDirection, YDigitalIO 377
get_portOpenDrain, YDigitalIO 378
get_portPolarity, YDigitalIO 379
get_portSize, YDigitalIO 380
get_portState, YDigitalIO 381
get_portState, YHubPort 670
get_position, YServo 1360
get_positionAtPowerOn, YServo 1361
get_power, YLed 742
get_powerControl, YDualPower 502
get_powerDuration, YWakeUpMonitor 1607
get_powerMode, YPwmPowerSource 1132
get_powerState, YDualPower 503
get_preview, YDataSet 333
get_primaryDNS, YNetwork 926
get_productId, YModule 873
get_productName, YModule 874
get_productRelease, YModule 875
get_progress, YDataSet 334
get_pulseCounter, YAnButton 100
get_pulseDuration, YPwmOutput 1098
get_pulseTimer, YAnButton 101
get_pulseTimer, YRelay 1279
get_pulseTimer, YWatchdog 1692
get_quaternionW, YGyro 622
get_quaternionX, YGyro 623
get_quaternionY, YGyro 624
get_quaternionZ, YGyro 625
get_range, YServo 1362
get_rawValue, YAnButton 102
get_readiness, YNetwork 927
get_rebootCountdown, YModule 876
get_recordedData, YAccelerometer 53
get_recordedData, YCarbonDioxide 141
get_recordedData, YCompass 218
get_recordedData, YCurrent 261
get_recordedData, YGenericSensor 570
get_recordedData, YGyro 626
get_recordedData, YHumidity 704
get_recordedData, YLightSensor 779
get_recordedData, YMagnetometer 822
get_recordedData, YPower 1010
get_recordedData, YPressure 1054
get_recordedData, YQt 1166
get_recordedData, YSensor 1320
get_recordedData, YTemperature 1402
get_recordedData, YTilt 1447
get_recordedData, YVoc 1490
get_recordedData, YVoltage 1533
get_recording, YDataLogger 302
get_regulationFailure, YVSource 1574
get_reportFrequency, YAccelerometer 54
get_reportFrequency, YCarbonDioxide 142
get_reportFrequency, YCompass 219
get_reportFrequency, YCurrent 262
get_reportFrequency, YGenericSensor 571
get_reportFrequency, YGyro 627
get_reportFrequency, YHumidity 705
get_reportFrequency, YLightSensor 780
get_reportFrequency, YMagnetometer 823
get_reportFrequency, YPower 1011
get_reportFrequency, YPressure 1055
get_reportFrequency, YQt 1167
get_reportFrequency, YSensor 1321
get_reportFrequency, YTemperature 1403
get_reportFrequency, YTilt 1448
get_reportFrequency, YVoc 1491
get_reportFrequency, YVoltage 1534
get_resolution, YAccelerometer 55
get_resolution, YCarbonDioxide 143
get_resolution, YCompass 220
get_resolution, YCurrent 263
get_resolution, YGenericSensor 572
get_resolution, YGyro 628
get_resolution, YHumidity 706
get_resolution, YLightSensor 781
get_resolution, YMagnetometer 824
get_resolution, YPower 1012
get_resolution, YPressure 1056
get_resolution, YQt 1168
get_resolution, YSensor 1322
get_resolution, YTemperature 1404
get_resolution, YTilt 1449
get_resolution, YVoc 1492
get_resolution, YVoltage 1535
get_rgbColor, YColorLed 178
get_rgbColorAtPowerOn, YColorLed 179
get_roll, YGyro 629
get_router, YNetwork 928
get_rowCount, YDataStream 351
get_runIndex, YDataStream 352
get_running, YWatchdog 1693
get_secondaryDNS, YNetwork 929
get_security, YWireless 1739
get_sensitivity, YAnButton 103
get_sensorType, YTemperature 1405
get_serialNumber, YModule 877
get_shutdownCountdown, YOsControl 973
get_signalRange, YGenericSensor 573
get_signalUnit, YGenericSensor 574
get_signalValue, YGenericSensor 575
get_sleepCountdown, YWakeUpMonitor 1608
get_ssid, YWireless 1740
get_startTime, YDataStream 353
get_startTimeUTC, YDataRun 323
get_startTimeUTC, YDataSet 335
get_startTimeUTC, YDataStream 354
get_startTimeUTC, YMeasure 851
get_startupSeq, YDisplay 431
get_state, YRelay 1280
get_state, YWatchdog 1694
get_stateAtPowerOn, YRelay 1281
get_stateAtPowerOn, YWatchdog 1695
get_subnetMask, YNetwork 930

get_summary, YDataSet 336
get_timeSet, YRealTimeClock 1203
get_timeUTC, YDataLogger 303
get_triggerDelay, YWatchdog 1696
get_triggerDuration, YWatchdog 1697
get_unit, YAccelerometer 56
get_unit, YCarbonDioxide 144
get_unit, YCompass 221
get_unit, YCurrent 264
get_unit, YDataSet 337
get_unit, YGenericSensor 576
get_unit, YGyro 630
get_unit, YHumidity 707
get_unit, YLightSensor 782
get_unit, YMagnetometer 825
get_unit, YPower 1013
get_unit, YPressure 1057
get_unit, YQt 1169
get_unit, YSensor 1323
get_unit, YTemperature 1406
get_unit, YTilt 1450
get_unit, YVoc 1493
get_unit, YVoltage 1536
get_unit, YVSource 1575
get_unixTime, YRealTimeClock 1204
get_upTime, YModule 878
get_usbBandwidth, YModule 879
get_usbCurrent, YModule 880
get_userData, YAccelerometer 57
get_userData, YAnButton 104
get_userData, YCarbonDioxide 145
get_userData, YColorLed 180
get_userData, YCompass 222
get_userData, YCurrent 265
get_userData, YDataLogger 304
get_userData, YDigitalIO 382
get_userData, YDisplay 432
get_userData, YDualPower 504
get_userData, YFiles 536
get_userData, YGenericSensor 577
get_userData, YGyro 631
get_userData, YHubPort 671
get_userData, YHumidity 708
get_userData, YLed 743
get_userData, YLightSensor 783
get_userData, YMagnetometer 826
get_userData, YModule 881
get_userData, YNetwork 931
get_userData, YOsControl 974
get_userData, YPower 1014
get_userData, YPressure 1058
get_userData, YPwmOutput 1099
get_userData, YPwmPowerSource 1133
get_userData, YQt 1170
get_userData, YRealTimeClock 1205
get_userData, YRefFrame 1243
get_userData, YRelay 1282
get_userData, YSensor 1324
get_userData, YServo 1363

get_userData, YTemperature 1407
get_userData, YTilt 1451
get_userData, YVoc 1494
get_userData, YVoltage 1537
get_userData, YVSource 1576
get_userData, YWakeUpMonitor 1609
get_userData, YWakeUpSchedule 1652
get_userData, YWatchdog 1698
get_userData, YWireless 1741
get_userPassword, YNetwork 932
get_utcOffset, YRealTimeClock 1206
get_valueCount, YDataRun 324
get_valueInterval, YDataRun 325
get_valueRange, YGenericSensor 578
get_voltage, YVSource 1577
get_wakeUpReason, YWakeUpMonitor 1610
get_wakeUpState, YWakeUpMonitor 1611
get_weekDays, YWakeUpSchedule 1653
get_wwwWatchdogDelay, YNetwork 933
get_xValue, YAccelerometer 58
get_xValue, YGyro 632
get_xValue, YMagnetometer 827
get_yValue, YAccelerometer 59
get_yValue, YGyro 633
get_yValue, YMagnetometer 828
get_zValue, YAccelerometer 60
get_zValue, YGyro 634
get_zValue, YMagnetometer 829
GetAPIVersion, YAPI 18
GetTickCount, YAPI 19
Gyro 599

H

HandleEvents, YAPI 20
hide, YDisplayLayer 472
Horloge 1188
hslMove, YColorLed 181
Humidity 683

I

InitAPI, YAPI 21
Interface 32, 78, 120, 163, 196, 240, 283, 356,
404, 455, 487, 516, 549, 599, 654, 683, 726,
757, 801, 853, 898, 986, 1033, 1076, 1118,
1145, 1188, 1259, 1299, 1342, 1381, 1426,
1469, 1512, 1555, 1591, 1630, 1671, 1720
Introduction 1
isOnline, YAccelerometer 61
isOnline, YAnButton 105
isOnline, YCarbonDioxide 146
isOnline, YColorLed 182
isOnline, YCompass 223
isOnline, YCurrent 266
isOnline, YDataLogger 305
isOnline, YDigitalIO 383
isOnline, YDisplay 433
isOnline, YDualPower 505
isOnline, YFiles 537

isOnline, YGenericSensor 579
isOnline, YGyro 635
isOnline, YHubPort 672
isOnline, YHumidity 709
isOnline, YLed 744
isOnline, YLightSensor 784
isOnline, YMagnetometer 830
isOnline, YModule 882
isOnline, YNetwork 934
isOnline, YOsControl 975
isOnline, YPower 1015
isOnline, YPressure 1059
isOnline, YPwmOutput 1100
isOnline, YPwmPowerSource 1134
isOnline, YQt 1171
isOnline, YRealTimeClock 1207
isOnline, YRefFrame 1244
isOnline, YRelay 1283
isOnline, YSensor 1325
isOnline, YServo 1364
isOnline, YTemperature 1408
isOnline, YTilt 1452
isOnline, YVoc 1495
isOnline, YVoltage 1538
isOnline, YVSource 1578
isOnline, YWakeUpMonitor 1612
isOnline, YWakeUpSchedule 1654
isOnline, YWatchdog 1699
isOnline, YWireless 1742
isOnline_async, YAccelerometer 62
isOnline_async, YAnButton 106
isOnline_async, YCarbonDioxide 147
isOnline_async, YColorLed 183
isOnline_async, YCompass 224
isOnline_async, YCurrent 267
isOnline_async, YDataLogger 306
isOnline_async, YDigitalIO 384
isOnline_async, YDisplay 434
isOnline_async, YDualPower 506
isOnline_async, YFiles 538
isOnline_async, YGenericSensor 580
isOnline_async, YGyro 636
isOnline_async, YHubPort 673
isOnline_async, YHumidity 710
isOnline_async, YLed 745
isOnline_async, YLightSensor 785
isOnline_async, YMagnetometer 831
isOnline_async, YModule 883
isOnline_async, YNetwork 935
isOnline_async, YOsControl 976
isOnline_async, YPower 1016
isOnline_async, YPressure 1060
isOnline_async, YPwmOutput 1101
isOnline_async, YPwmPowerSource 1135
isOnline_async, YQt 1172
isOnline_async, YRealTimeClock 1208
isOnline_async, YRefFrame 1245
isOnline_async, YRelay 1284
isOnline_async, YSensor 1326

isOnline_async, YServo 1365
isOnline_async, YTemperature 1409
isOnline_async, YTilt 1453
isOnline_async, YVoc 1496
isOnline_async, YVoltage 1539
isOnline_async, YVSource 1579
isOnline_async, YWakeUpMonitor 1613
isOnline_async, YWakeUpSchedule 1655
isOnline_async, YWatchdog 1700
isOnline_async, YWireless 1743

J

Javascript 3
joinNetwork, YWireless 1744

L

LightSensor 757
lineTo, YDisplayLayer 473
load, YAccelerometer 63
load, YAnButton 107
load, YCarbonDioxide 148
load, YColorLed 184
load, YCompass 225
load, YCurrent 268
load, YDataLogger 307
load, YDigitalIO 385
load, YDisplay 435
load, YDualPower 507
load, YFiles 539
load, YGenericSensor 581
load, YGyro 637
load, YHubPort 674
load, YHumidity 711
load, YLed 746
load, YLightSensor 786
load, YMagnetometer 832
load, YModule 884
load, YNetwork 936
load, YOsControl 977
load, YPower 1017
load, YPressure 1061
load, YPwmOutput 1102
load, YPwmPowerSource 1136
load, YQt 1173
load, YRealTimeClock 1209
load, YRefFrame 1246
load, YRelay 1285
load, YSensor 1327
load, YServo 1366
load, YTemperature 1410
load, YTilt 1454
load, YVoc 1497
load, YVoltage 1540
load, YVSource 1580
load, YWakeUpMonitor 1614
load, YWakeUpSchedule 1656
load, YWatchdog 1701
load, YWireless 1745

load_async, YAccelerometer 65
load_async, YAnButton 108
load_async, YCarbonDioxide 150
load_async, YColorLed 185
load_async, YCompass 227
load_async, YCurrent 270
load_async, YDataLogger 308
load_async, YDigitalIO 386
load_async, YDisplay 436
load_async, YDualPower 508
load_async, YFiles 540
load_async, YGenericSensor 583
load_async, YGyro 639
load_async, YHubPort 675
load_async, YHumidity 713
load_async, YLed 747
load_async, YLightSensor 788
load_async, YMagnetometer 834
load_async, YModule 885
load_async, YNetwork 937
load_async, YOsControl 978
load_async, YPower 1019
load_async, YPressure 1063
load_async, YPwmOutput 1103
load_async, YPwmPowerSource 1137
load_async, YQt 1175
load_async, YRealTimeClock 1210
load_async, YRefFrame 1247
load_async, YRelay 1286
load_async, YSensor 1329
load_async, YServo 1367
load_async, YTemperature 1412
load_async, YTilt 1456
load_async, YVoc 1499
load_async, YVoltage 1542
load_async, YVSource 1581
load_async, YWakeUpMonitor 1615
load_async, YWakeUpSchedule 1657
load_async, YWatchdog 1702
load_async, YWireless 1746
loadCalibrationPoints, YAccelerometer 64
loadCalibrationPoints, YCarbonDioxide 149
loadCalibrationPoints, YCompass 226
loadCalibrationPoints, YCurrent 269
loadCalibrationPoints, YGenericSensor 582
loadCalibrationPoints, YGyro 638
loadCalibrationPoints, YHumidity 712
loadCalibrationPoints, YLightSensor 787
loadCalibrationPoints, YMagnetometer 833
loadCalibrationPoints, YPower 1018
loadCalibrationPoints, YPressure 1062
loadCalibrationPoints, YQt 1174
loadCalibrationPoints, YSensor 1328
loadCalibrationPoints, YTemperature 1411
loadCalibrationPoints, YTilt 1455
loadCalibrationPoints, YVoc 1498
loadCalibrationPoints, YVoltage 1541
loadMore, YDataSet 338
loadMore_async, YDataSet 339

M

Magnetometer 801
Mesurée 847
Mise 318
Module 5, 853
more3DCalibration, YRefFrame 1248
move, YServo 1368
moveTo, YDisplayLayer 474

N

Network 898
newSequence, YDisplay 437
nextAccelerometer, YAccelerometer 66
nextAnButton, YAnButton 109
nextCarbonDioxide, YCarbonDioxide 151
nextColorLed, YColorLed 186
nextCompass, YCompass 228
nextCurrent, YCurrent 271
nextDataLogger, YDataLogger 309
nextDigitalIO, YDigitalIO 387
nextDisplay, YDisplay 438
nextDualPower, YDualPower 509
nextFiles, YFiles 541
nextGenericSensor, YGenericSensor 584
nextGyro, YGyro 640
nextHubPort, YHubPort 676
nextHumidity, YHumidity 714
nextLed, YLed 748
nextLightSensor, YLightSensor 789
nextMagnetometer, YMagnetometer 835
nextModule, YModule 886
nextNetwork, YNetwork 938
nextOsControl, YOsControl 979
nextPower, YPower 1020
nextPressure, YPressure 1064
nextPwmOutput, YPwmOutput 1104
nextPwmPowerSource, YPwmPowerSource 1138
nextQt, YQt 1176
nextRealTimeClock, YRealTimeClock 1211
nextRefFrame, YRefFrame 1249
nextRelay, YRelay 1287
nextSensor, YSensor 1330
nextServo, YServo 1369
nextTemperature, YTemperature 1413
nextTilt, YTilt 1457
nextVoc, YVoc 1500
nextVoltage, YVoltage 1543
nextVSource, YVSource 1582
nextWakeUpMonitor, YWakeUpMonitor 1616
nextWakeUpSchedule, YWakeUpSchedule 1658
nextWatchdog, YWatchdog 1703
nextWireless, YWireless 1747

O

Objets 455

P

pauseSequence, YDisplay 439
ping, YNetwork 939
playSequence, YDisplay 440
Port 654
Power 986
Préparation 3
PreregisterHub, YAPI 22
Pressure 1033
pulse, YDigitalIO 388
pulse, YRelay 1288
pulse, YVSource 1583
pulse, YWatchdog 1704
pulseDurationMove, YPwmOutput 1105
PwmPowerSource 1118

Q

Quaternion 1145

R

Real 1188
reboot, YModule 887
Reference 12
Référentiel 1219
registerAnglesCallback, YGyro 641
RegisterDeviceArrivalCallback, YAPI 23
RegisterDeviceRemovalCallback, YAPI 24
RegisterHub, YAPI 25
registerQuaternionCallback, YGyro 642
registerTimedReportCallback, YAccelerometer 67
registerTimedReportCallback, YCarbonDioxide 152
registerTimedReportCallback, YCompass 229
registerTimedReportCallback, YCurrent 272
registerTimedReportCallback, YGenericSensor 585
registerTimedReportCallback, YGyro 643
registerTimedReportCallback, YHumidity 715
registerTimedReportCallback, YLightSensor 790
registerTimedReportCallback, YMagnetometer 836
registerTimedReportCallback, YPower 1021
registerTimedReportCallback, YPressure 1065
registerTimedReportCallback, YQt 1177
registerTimedReportCallback, YSensor 1331
registerTimedReportCallback, YTemperature 1414
registerTimedReportCallback, YTilt 1458
registerTimedReportCallback, YVoc 1501
registerTimedReportCallback, YVoltage 1544
registerValueCallback, YAccelerometer 68
registerValueCallback, YAnButton 110
registerValueCallback, YCarbonDioxide 153
registerValueCallback, YColorLed 187
registerValueCallback, YCompass 230
registerValueCallback, YCurrent 273

registerValueCallback, YDataLogger 310
registerValueCallback, YDigitalIO 389
registerValueCallback, YDisplay 441
registerValueCallback, YDualPower 510
registerValueCallback, YFiles 542
registerValueCallback, YGenericSensor 586
registerValueCallback, YGyro 644
registerValueCallback, YHubPort 677
registerValueCallback, YHumidity 716
registerValueCallback, YLed 749
registerValueCallback, YLightSensor 791
registerValueCallback, YMagnetometer 837
registerValueCallback, YNetwork 940
registerValueCallback, YOsControl 980
registerValueCallback, YPower 1022
registerValueCallback, YPressure 1066
registerValueCallback, YPwmOutput 1106
registerValueCallback, YPwmPowerSource 1139
registerValueCallback, YQt 1178
registerValueCallback, YRealTimeClock 1212
registerValueCallback, YRefFrame 1250
registerValueCallback, YRelay 1289
registerValueCallback, YSensor 1332
registerValueCallback, YServo 1370
registerValueCallback, YTemperature 1415
registerValueCallback, YTilt 1459
registerValueCallback, YVoc 1502
registerValueCallback, YVoltage 1545
registerValueCallback, YVSource 1584
registerValueCallback, YWakeUpMonitor 1617
registerValueCallback, YWakeUpSchedule 1659
registerValueCallback, YWatchdog 1705
registerValueCallback, YWireless 1748
Relay 1259
remove, YFiles 543
reset, YDisplayLayer 475
reset, YPower 1023
resetAll, YDisplay 442
resetCounter, YAnButton 111
resetSleepCountDown, YWakeUpMonitor 1618
resetWatchdog, YWatchdog 1706
revertFromFlash, YModule 888
rgbMove, YColorLed 188

S

save3DCalibration, YRefFrame 1251
saveSequence, YDisplay 443
saveToFlash, YModule 889
selectColorPen, YDisplayLayer 476
selectEraser, YDisplayLayer 477
selectFont, YDisplayLayer 478
selectGrayPen, YDisplayLayer 479
Senseur 1299
Séquence 318, 328, 341
Servo 1342
set_adminPassword, YNetwork 941
set_analogCalibration, YAnButton 112
set_autoStart, YDataLogger 311
set_autoStart, YWatchdog 1707

set_beacon, YModule 890
set_bearing, YRefFrame 1252
set_bitDirection, YDigitalIO 390
set_bitOpenDrain, YDigitalIO 391
set_bitPolarity, YDigitalIO 392
set_bitState, YDigitalIO 393
set_blinking, YLed 750
set_brightness, YDisplay 444
set_calibrationMax, YAnButton 113
set_calibrationMin, YAnButton 114
set_callbackCredentials, YNetwork 942
set_callbackEncoding, YNetwork 943
set_callbackMaxDelay, YNetwork 944
set_callbackMethod, YNetwork 945
set_callbackMinDelay, YNetwork 946
set_callbackUrl, YNetwork 947
set_discoverable, YNetwork 948
set_dutyCycle, YPwmOutput 1107
set_dutyCycleAtPowerOn, YPwmOutput 1108
set_enabled, YDisplay 445
set_enabled, YHubPort 678
set_enabled, YPwmOutput 1109
set_enabled, YServo 1371
set_enabledAtPowerOn, YPwmOutput 1110
set_enabledAtPowerOn, YServo 1372
set_frequency, YPwmOutput 1111
set_highestValue, YAccelerometer 69
set_highestValue, YCarbonDioxide 154
set_highestValue, YCompass 231
set_highestValue, YCurrent 274
set_highestValue, YGenericSensor 587
set_highestValue, YGyro 645
set_highestValue, YHumidity 717
set_highestValue, YLightSensor 792
set_highestValue, YMagnetometer 838
set_highestValue, YPower 1024
set_highestValue, YPressure 1067
set_highestValue, YQt 1179
set_highestValue, YSensor 1333
set_highestValue, YTemperature 1416
set_highestValue, YTilt 1460
set_highestValue, YVoc 1503
set_highestValue, YVoltage 1546
set_hours, YWakeUpSchedule 1660
set_hslColor, YColorLed 189
set_logFrequency, YAccelerometer 70
set_logFrequency, YCarbonDioxide 155
set_logFrequency, YCompass 232
set_logFrequency, YCurrent 275
set_logFrequency, YGenericSensor 588
set_logFrequency, YGyro 646
set_logFrequency, YHumidity 718
set_logFrequency, YLightSensor 793
set_logFrequency, YMagnetometer 839
set_logFrequency, YPower 1025
set_logFrequency, YPressure 1068
set_logFrequency, YQt 1180
set_logFrequency, YSensor 1334
set_logFrequency, YTemperature 1417
set_logFrequency, YTilt 1461
set_logFrequency, YVoc 1504
set_logFrequency, YVoltage 1547
set_logicalName, YAccelerometer 71
set_logicalName, YAnButton 115
set_logicalName, YCarbonDioxide 156
set_logicalName, YColorLed 190
set_logicalName, YCompass 233
set_logicalName, YCurrent 276
set_logicalName, YDataLogger 312
set_logicalName, YDigitalIO 394
set_logicalName, YDisplay 446
set_logicalName, YDualPower 511
set_logicalName, YFiles 544
set_logicalName, YGenericSensor 589
set_logicalName, YGyro 647
set_logicalName, YHubPort 679
set_logicalName, YHumidity 719
set_logicalName, YLed 751
set_logicalName, YLightSensor 794
set_logicalName, YMagnetometer 840
set_logicalName, YModule 891
set_logicalName, YNetwork 949
set_logicalName, YOsControl 981
set_logicalName, YPower 1026
set_logicalName, YPressure 1069
set_logicalName, YPwmOutput 1112
set_logicalName, YPwmPowerSource 1140
set_logicalName, YQt 1181
set_logicalName, YRealTimeClock 1213
set_logicalName, YRefFrame 1253
set_logicalName, YRelay 1290
set_logicalName, YSensor 1335
set_logicalName, YServo 1373
set_logicalName, YTemperature 1418
set_logicalName, YTilt 1462
set_logicalName, YVoc 1505
set_logicalName, YVoltage 1548
set_logicalName, YVSource 1585
set_logicalName, YWakeUpMonitor 1619
set_logicalName, YWakeUpSchedule 1661
set_logicalName, YWatchdog 1708
set_logicalName, YWireless 1749
set_lowestValue, YAccelerometer 72
set_lowestValue, YCarbonDioxide 157
set_lowestValue, YCompass 234
set_lowestValue, YCurrent 277
set_lowestValue, YGenericSensor 590
set_lowestValue, YGyro 648
set_lowestValue, YHumidity 720
set_lowestValue, YLightSensor 795
set_lowestValue, YMagnetometer 841
set_lowestValue, YPower 1027
set_lowestValue, YPressure 1070
set_lowestValue, YQt 1182
set_lowestValue, YSensor 1336
set_lowestValue, YTemperature 1419
set_lowestValue, YTilt 1463
set_lowestValue, YVoc 1506

set_lowestValue, YVoltage 1549
set_luminosity, YLed 752
set_luminosity, YModule 892
set_maxTimeOnStateA, YRelay 1291
set_maxTimeOnStateA, YWatchdog 1709
set_maxTimeOnStateB, YRelay 1292
set_maxTimeOnStateB, YWatchdog 1710
set_minutes, YWakeUpSchedule 1662
set_minutesA, YWakeUpSchedule 1663
set_minutesB, YWakeUpSchedule 1664
set_monthDays, YWakeUpSchedule 1665
set_months, YWakeUpSchedule 1666
set_mountPosition, YRefFrame 1254
set_neutral, YServo 1374
set_nextWakeUp, YWakeUpMonitor 1620
set_orientation, YDisplay 447
set_output, YRelay 1293
set_output, YWatchdog 1711
set_outputVoltage, YDigitalIO 395
set_period, YPwmOutput 1113
set_portDirection, YDigitalIO 396
set_portOpenDrain, YDigitalIO 397
set_portPolarity, YDigitalIO 398
set_portState, YDigitalIO 399
set_position, YServo 1375
set_positionAtPowerOn, YServo 1376
set_power, YLed 753
set_powerControl, YDualPower 512
set_powerDuration, YWakeUpMonitor 1621
set_powerMode, YPwmPowerSource 1141
set_primaryDNS, YNetwork 950
set_pulseDuration, YPwmOutput 1114
set_range, YServo 1377
set_recording, YDataLogger 313
set_reportFrequency, YAccelerometer 73
set_reportFrequency, YCarbonDioxide 158
set_reportFrequency, YCompass 235
set_reportFrequency, YCurrent 278
set_reportFrequency, YGenericSensor 591
set_reportFrequency, YGyro 649
set_reportFrequency, YHumidity 721
set_reportFrequency, YLightSensor 796
set_reportFrequency, YMagnetometer 842
set_reportFrequency, YPower 1028
set_reportFrequency, YPressure 1071
set_reportFrequency, YQt 1183
set_reportFrequency, YSensor 1337
set_reportFrequency, YTemperature 1420
set_reportFrequency, YTilt 1464
set_reportFrequency, YVoc 1507
set_reportFrequency, YVoltage 1550
set_resolution, YAccelerometer 74
set_resolution, YCarbonDioxide 159
set_resolution, YCompass 236
set_resolution, YCurrent 279
set_resolution, YGenericSensor 592
set_resolution, YGyro 650
set_resolution, YHumidity 722
set_resolution, YLightSensor 797
set_resolution, YMagnetometer 843
set_resolution, YPower 1029
set_resolution, YPressure 1072
set_resolution, YQt 1184
set_resolution, YSensor 1338
set_resolution, YTemperature 1421
set_resolution, YTilt 1465
set_resolution, YVoc 1508
set_resolution, YVoltage 1551
set_rgbColor, YColorLed 191
set_rgbColorAtPowerOn, YColorLed 192
set_running, YWatchdog 1712
set_secondaryDNS, YNetwork 951
set_sensitivity, YAnButton 116
set_sensorType, YTemperature 1422
set_signalRange, YGenericSensor 593
set_sleepCountdown, YWakeUpMonitor 1622
set_startupSeq, YDisplay 448
set_state, YRelay 1294
set_state, YWatchdog 1713
set_stateAtPowerOn, YRelay 1295
set_stateAtPowerOn, YWatchdog 1714
set_timeUTC, YDataLogger 314
set_triggerDelay, YWatchdog 1715
set_triggerDuration, YWatchdog 1716
set_unit, YGenericSensor 594
set_unixTime, YRealTimeClock 1214
set_usbBandwidth, YModule 893
set_userData, YAccelerometer 75
set_userData, YAnButton 117
set_userData, YCarbonDioxide 160
set_userData, YColorLed 193
set_userData, YCompass 237
set_userData, YCurrent 280
set_userData, YDataLogger 315
set_userData, YDigitalIO 400
set_userData, YDisplay 449
set_userData, YDualPower 513
set_userData, YFiles 545
set_userData, YGenericSensor 595
set_userData, YGyro 651
set_userData, YHubPort 680
set_userData, YHumidity 723
set_userData, YLed 754
set_userData, YLightSensor 798
set_userData, YMagnetometer 844
set_userData, YModule 894
set_userData, YNetwork 952
set_userData, YOsControl 982
set_userData, YPower 1030
set_userData, YPressure 1073
set_userData, YPwmOutput 1115
set_userData, YPwmPowerSource 1142
set_userData, YQt 1185
set_userData, YRealTimeClock 1215
set_userData, YRefFrame 1255
set_userData, YRelay 1296
set_userData, YSensor 1339
set_userData, YServo 1378

set_userdata, YTemperature 1423
set_userdata, YTilt 1466
set_userdata, YVoc 1509
set_userdata, YVoltage 1552
set_userdata, YVSource 1586
set_userdata, YWakeUpMonitor 1623
set_userdata, YWakeUpSchedule 1667
set_userdata, YWatchdog 1717
set_userdata, YWireless 1750
set_userPassword, YNetwork 953
set_utcOffset, YRealTimeClock 1216
set_valueInterval, YDataRun 326
set_valueRange, YGenericSensor 596
set_voltage, YVSource 1587
set_weekDays, YWakeUpSchedule 1668
set_wwwWatchdogDelay, YNetwork 954
setAntialiasingMode, YDisplayLayer 480
setConsoleBackground, YDisplayLayer 481
setConsoleMargins, YDisplayLayer 482
setConsoleWordWrap, YDisplayLayer 483
setLayerPosition, YDisplayLayer 484
SetTimeout, YAPI 26
shutdown, YOsControl 983
Sleep, YAPI 27
sleep, YWakeUpMonitor 1624
sleepFor, YWakeUpMonitor 1625
sleepUntil, YWakeUpMonitor 1626
Source 1555
start3DCalibration, YRefFrame 1256
stopSequence, YDisplay 450
swapLayerContent, YDisplay 451

T

Temperature 1381
Temps 1188
Tension 1555
Tilt 1426
toggle_bitState, YDigitalIO 401
triggerFirmwareUpdate, YModule 895
Type 1299

U

unhide, YDisplayLayer 485
UnregisterHub, YAPI 28
UpdateDeviceList, YAPI 29
UpdateDeviceList_async, YAPI 30
upload, YDisplay 452
upload, YFiles 546
useDHCP, YNetwork 955
useStaticIP, YNetwork 956

V

Valeur 847
Voltage 1512
voltageMove, YVSource 1588

W

wait_async, YAccelerometer 76
wait_async, YAnButton 118
wait_async, YCarbonDioxide 161
wait_async, YColorLed 194
wait_async, YCompass 238
wait_async, YCurrent 281
wait_async, YDataLogger 316
wait_async, YDigitalIO 402
wait_async, YDisplay 453
wait_async, YDualPower 514
wait_async, YFiles 547
wait_async, YGenericSensor 597
wait_async, YGyro 652
wait_async, YHubPort 681
wait_async, YHumidity 724
wait_async, YLed 755
wait_async, YLightSensor 799
wait_async, YMagnetometer 845
wait_async, YModule 896
wait_async, YNetwork 957
wait_async, YOsControl 984
wait_async, YPower 1031
wait_async, YPressure 1074
wait_async, YPwmOutput 1116
wait_async, YPwmPowerSource 1143
wait_async, YQt 1186
wait_async, YRealTimeClock 1217
wait_async, YRefFrame 1257
wait_async, YRelay 1297
wait_async, YSensor 1340
wait_async, YServo 1379
wait_async, YTemperature 1424
wait_async, YTilt 1467
wait_async, YVoc 1510
wait_async, YVoltage 1553
wait_async, YVSource 1589
wait_async, YWakeUpMonitor 1627
wait_async, YWakeUpSchedule 1669
wait_async, YWatchdog 1718
wait_async, YWireless 1751
wakeUp, YWakeUpMonitor 1628
WakeUpMonitor 1591
WakeUpSchedule 1630
Watchdog 1671
Wireless 1720

Y

YAccelerometer 34-76
YAnButton 80-118
YAPI 14-30
YCarbonDioxide 122-161
yCheckLogicalName 14
YColorLed 164-194
YCompass 198-238
YCurrent 242-281
YDataLogger 284-316

YDataRun 318-326
YDataSet 329-339
YDataStream 342-354
YDigitalIO 358-402
yDisableExceptions 15
YDisplay 406-453
YDisplayLayer 456-485
YDualPower 488-514
yEnableExceptions 16
YFiles 517-547
yFindAccelerometer 34
yFindAnButton 80
yFindCarbonDioxide 122
yFindColorLed 164
yFindCompass 198
yFindCurrent 242
yFindDataLogger 284
yFindDigitalIO 358
yFindDisplay 406
yFindDualPower 488
yFindFiles 517
yFindGenericSensor 551
yFindGyro 601
yFindHubPort 655
yFindHumidity 685
yFindLed 727
yFindLightSensor 759
yFindMagnetometer 803
yFindModule 855
yFindNetwork 901
yFindOsControl 960
yFindPower 988
yFindPressure 1035
yFindPwmOutput 1078
yFindPwmPowerSource 1119
yFindQt 1147
yFindRealTimeClock 1189
yFindRefFrame 1221
yFindRelay 1261
yFindSensor 1302
yFindServo 1345
yFindTemperature 1383
yFindTilt 1429
yFindVoc 1472
yFindVoltage 1515
yFindVSource 1557
yFindWakeUpMonitor 1594
yFindWakeUpSchedule 1632
yFindWatchdog 1674
yFindWireless 1722
yFirstAccelerometer 35
yFirstAnButton 81
yFirstCarbonDioxide 123
yFirstColorLed 165
yFirstCompass 199
yFirstCurrent 243
yFirstDataLogger 285
yFirstDigitalIO 359
yFirstDisplay 407
yFirstDualPower 489
yFirstFiles 518
yFirstGenericSensor 552
yFirstGyro 602
yFirstHubPort 656
yFirstHumidity 686
yFirstLed 728
yFirstLightSensor 760
yFirstMagnetometer 804
yFirstModule 856
yFirstNetwork 902
yFirstOsControl 961
yFirstPower 989
yFirstPressure 1036
yFirstPwmOutput 1079
yFirstPwmPowerSource 1120
yFirstQt 1148
yFirstRealTimeClock 1190
yFirstRefFrame 1222
yFirstRelay 1262
yFirstSensor 1302
yFirstServo 1345
yFirstTemperature 1384
yFirstTilt 1429
yFirstVoc 1472
yFirstVoltage 1515
yFirstVSource 1557
yFirstWakeUpMonitor 1594
yFirstWakeUpSchedule 1633
yFirstWatchdog 1674
yFirstWireless 1722
yFreeAPI 17
YGenericSensor 551-597
yGetAPIVersion 18
yGetTickCount 19
YGyro 601-652
yHandleEvents 20
YHubPort 655-681
YHumidity 685-724
yInitAPI 21
YLed 727-755
YLightSensor 759-799
YMagnetometer 803-845
YMeasure 847-851
YModule 855-896
YNetwork 901-957
Yocto-Demo 3
Yocto-hub 654
YOsControl 960-984
YPower 988-1031
yPreregisterHub 22
YPressure 1035-1074
YPwmOutput 1078-1116
YPwmPowerSource 1119-1143
YQt 1147-1186
YRealTimeClock 1189-1217
YRefFrame 1221-1257
yRegisterDeviceArrivalCallback 23
yRegisterDeviceRemovalCallback 24

yRegisterHub 25
YRelay 1261-1297
YSensor 1301-1340
YServo 1344-1379
ySetTimeout 26
ySleep 27
YTemperature 1383-1424
YTilt 1428-1467
yUnregisterHub 28

yUpdateDeviceList 29
yUpdateDeviceList_async 30
YVoc 1471-1510
YVoltage 1514-1553
YVSource 1556-1589
YWakeUpMonitor 1593-1628
YWakeUpSchedule 1632-1669
YWatchdog 1673-1718
YWireless 1721-1751